SOCS

A COMPUTATIONAL LOGIC MODEL FOR THE DESCRIPTION, ANALYSIS AND VERIFICATION OF GLOBAL AND OPEN SOCIETIES OF HETEROGENEOUS COMPUTEES

IST-2001-32530

Examples of the functioning of computees and their societies

Project number:	IST-2001-32530
Project acronym:	SOCS
Document type:	D (deliverable)
Document distribution:	I (internal to SOCS and PO)
CEC Document number:	IST32530/ICSTM/33/D/I/b1
File name:	133-b1[examples].ps
Editor:	Kostas Stathis and Francesca Toni
Contributing partners:	ALL
Contributing workpackages:	WP3,WP4
Estimated person months:	4
Date of completion:	26 January 2004
Date of delivery to the EC:	31 January 2004
Number of pages:	47

ABSTRACT

This document identifies a number of examples of the models proposed in D4 and D5 for computees and their societies, respectively, as well as their integration. We have used the examples to test and adjust the suitability of the models in the context of the (success) criteria for self-assessment and evaluation put forward in D3. We have also used (some of) these examples to guide the development and testing of the prototype demonstrator within WP4. We use (some of) these examples in the demo deliverable D9.

Examples of the functioning of computees and their societies

Marco Alberti, Andrea Bracciali, Federico Chesani, Neophytous Demetriou, Ulle Endriss, Antonis Kakas, Evelina Lamma, Wenjin Lu, Paola Mello, Michela Milano, Fariba Sadri, Kostas Stathis, Francesca Toni, Paolo Torroni

ABSTRACT
This document identifies a number of examples of the models proposed in D4 and D5 for computees and their societies, respectively, as well as their integration. We have used the examples to test and adjust the suitability of the models in the context of the (success) criteria for self-assessment and evaluation put forward in D3. We have also used (some of) these examples to guide the development and testing of the prototype demonstrator within WP4. We use (some of) these examples in the demo deliverable D9.

Contents

1 Introduction

In D3 we have identified a number of criteria for assessing the various (technical) workpackages of the SOCS project. We have further classified these criteria in three categories.

- *Success criteria* (marked by + in D3) – These criteria indicate the minimum requirements for the project within a given workpackage. These criteria must be fulfilled in that they aim at pushing the state of the art as (minimally) required for the success of the project.

- *Extension criteria* (marked by + in D3) – This second kind of criteria indicates requirements that the project will aim at fulfilling with the same rigour as the first kind, if time and resources will allow, or in any case by carrying out some feasibility studies on their achievement, if insufficient time and unforeseen difficulties with the first type of criteria will arise.

- Future direction criteria (marked by * in D3) – The third kind of criteria indicates requirements that the project does not need to fulfil in order to be deemed successful, but that indicate interesting directions for future work. We will aim at fulfilling these criteria only if time and resources will allow.

In order to contextualise the implementation activity and link it with the criteria outlined in D3, in this document we identify examples for each of the success criteria for WP1 and WP2. These examples have been used during the development of the models, for the dynamic evaluation of the models with respect to the success criteria (see the evaluation methodology for WP1 and WP2 in D3), and to guide the development of the prototype demonstrator in Phase 2 of WP4 (see the evaluation methodology for WP4 in D3). Some of these examples are also used within deliverable D9 and are part of the prototype demonstrator for this deliverable.

The identified examples are presented in the context of the *Leaving San Vincenzo* scenario outlined in [Sta02]. However, the examples extend the scenario by specialising and augmenting further the interactions originally presented. We shall also assume for presentation purposes, that a piece of information is like a resource (or in some cases like a product) which can be exchanged between computees. By making this assumption we can treat interactions supporting the exchange of information viewed as resources similarly to the interactions often taking place in resource re-allocation problems (see [EMST02, STT02, TS02]), with the additional advantage of contextualising these interactions and present solutions that can be immediately useful to what people do in their everyday activities.

The structure of this document is as follows. First we describe the extensions, that we use to formulate all examples in this document, to the existing Leaving San Vincenzo scenario. We then present separately examples that test the behaviour of individual computees (WP1), of societies of computees (WP2), and of computees within societies (WP1 and WP2). Finally, we conclude.

2 The extended Leaving San Vincenzo scenario

We use the *Leaving San Vincenzo* scenario to provide the context of the examples discussed in this document. To establish the link of these examples to Global Computing (GC), we discuss the relevance of the Leaving San Vincenzo scenario to the GC programme. We also integrate within the San Vincenzo scenario the Resource Allocation [EMST02, STT02, TS02] and (Combinatorial) Auctions scenaria [MTG+03] proposed for SOCS, giving rise to a new and extended San Vincenzo scenario. The simplifying assumptions of the extensions are further discussed, however, only some of the identified extensions are outlined in this section, as most of the extensions are explained in detail in the sections that follow, in the form of concrete examples.

2.1 Summary of the *Leaving San Vincenzo* scenario

The original *Leaving San Vincenzo* scenario can be summarised as follows. A Spanish businessman, called Francisco Martinez, travels for work purposes to Italy and, in order to make his trip easier, carries a personal communicator, namely a device that is a hybrid between a mobile phone and a PDA. The application running on this personal communicator provides the environment for a computee, treated for the purposes of the scenario as a piece of software that augments the direct manipulation interface of the device with implicit management. By implicit management we mean that the computee is a personal service agent that provides proactive information management within the device [SdM02] and flexible connectivity to smart services available in the global environment the businessman travels within.

Most of the the original scenario unfolds in San Vincenzo, an Italian holiday resort, where Francisco spends the last weekend of his away trip. In this context, the scenario describes a series of events illustrating the kind of interactions (both physical and virtual) that Francisco's computee facilitates via the personal communicator of Francisco. Francisco's communicator connects Francisco's computee with a series of other devices that we assume are available in the physical environment so that Francisco can access smart electronic services. Here are the main examples used in the scenario:

- Before leaving Spain, at the airport gate the computee registers Francisco with a location-indpendent smart-service provider in Italy.
- Francisco arrives in Italy (Rome) during a weekend, and is uncertain whether to stay over or carry on to his destination. The computee advises Francisco to stay, as there is a jazz festival in Rome, that the computee believes to be of interest to Francisco.
- At the hotel Francisco stays during the last two days of his visit to Italy, in San Vincenzo, the computee pays Francisco's bill, orders a taxi, and checks for train information.
- At the train station of San Vincenzo, the computee advises Francisco how to successfully buy a train ticket, despite the difficult conditions (ticket office closed and ticket machine out of order).

Based on the above concrete examples, the scenario seeks to demonstrate how technology can be used to help people with what they do in their ordinary lives, from going to a business trip, to staying in a hotel, to travelling by train, to ordering a taxi.

2.2 Why is *Leaving San Vincenzo* relevant to GC?

GC has been proposed as a new vision for information and communication technologies whereby most physical objects that people use will be transformed into electronic devices with computing processors and embedded software designed to perform (or control) a multitude of tasks in people's everyday activities. Many of these devices will also be able to communicate with each other and to interact with the environment in carrying out the tasks they are designed for. The expectation is that the design of computational and information processing systems will rely entirely on the available infrastructure and processing power around us. The San Vincenzo scenario illustrates very effectively the huge challenges this vision poses in that it requires the definition and exploitation of dynamically configured applications of software entities interacting in novel ways with other entities to achieve or control their computational tasks for the benefit of ordinary people.

Another important feature of GC environments is that they are envisioned to be highly *open and dynamic*: physical devices will be mobile or portable, connectivity and bandwidth will not be constant, computational processes and data would need to migrate, and applications will come and go in the environment the same way people who use them will. The San Vincenzo scenario illustrates the openess of the global computing environment almost perfectly, as the context it is based upon is travelling. Indeed, in such unpredictable and open environments such as that outlined in Leaving San Vincenzo, one of the main issues is how to control the availability of the resources that are active in an application at any given time, which happens to be another important issue of the GC programme. By focusing SOCS on these issues, we aim at SOCS contributing to illustrate how to build the technologies of the future (required by GC) by using well-understood logical models and techniques.

2.3 Extending *Leaving San Vincenzo*

For the purposes of this document we extend the original San Vincenzo scenario in two ways: (a) we make Leaving San Vincenzo the "umbrella" scenario for SOCS by integrating into it the other two SOCS scenarios: the resource allocation scenario [EMST02, STT02, TS02] and the (combinatorial) auctions scenario [MTG$^+$03], and (b) we specialise and augment the interactions of the original scenario so that they can provide examples suitable for the implementation effort.

To achieve (a) we shall make the simplified assumption that a piece of information is like a resource (or in some cases like a product[CS01]) which can be exchanged between computees. By making this assumption we can treat interactions supporting the exchange of information similarly to the negotiations often taking place in resource allocation problems (in this way we incorporate in the San Vincenzo scenario the resource allocation scenario presented in the context of SOCS in [EMST02, STT02, TS02]), with the additional advantage of contextualising these interactions and present solutions that can be immediately useful to what people do in their everyday activities.

Also for (a) we shall interpret connectivity to smart services as *access to resources* available in and managed by societies of computees that "run" on top of a GC environment such as the one assumed by the scenario [TS02]. We also extend the original scenario to allow Francisco' computee to hold a (combinatorial) auction, e.g. in order to get a taxi at the lowest fare.

Finally (b), we will introduce novel, concrete examples of interactions in the Leaving San Vincenzo context as we go along. As we shall see, most of these new interactions are specialisations of the interactions presented in the original Leaving San Vincenzo scenario.

3 Examples for WP1: individual computees

In this section, we will exemplify the behaviour of Francisco's computee, as well as other computees that it needs to interact with, in a number of situations, to illustrate different aspects of the model of the individual computee as identified within deliverable D3. We will refer to Francisco's computee simply as f. For each aspect, we will identify a concrete instance of the initial state of f and/or of any other computee of interest (where the state of a computee consists of a triple $\langle KB, Goals, Plan \rangle$, as in D4) as well as their cycle theory $\mathcal{T}_{cycle}$ (again as in D4). Unless otherwise stated, in this section we will specify the state and behaviour of Francisco's computee f.

When giving the KB_{TR}, KB_{react} and KB_{plan} components of KB (of f or any other computee), we will only give the domain-dependent components of these knowledge bases, namely (and typically) the definitions of the predicates *holds_initially*, *initiates*, *terminates* and *precondition*, as well as any other definitions of interest. For the remaining, domain-independent, parts of KB_{TR}, KB_{react} and KB_{plan}, see deliverable D8. Moreover, unless differently specified, we will assume that $\mathcal{T}_{cycle} = \mathcal{T}_{initial} \cup \mathcal{T}_{basic} \cup \mathcal{T}_{interrupt} \cup \mathcal{T}_{behaviour}$ will be the *normal cycle theory* given in deliverable D8. This is summarised in appendix A. We will provide any domain-dependent definition in the auxiliary part of $\mathcal{T}_{cycle}$ explicitly, where required.
As in D4, we will assume that (timed) action operators are represented within the ($Plan$ component of the) state of a computee in the form

$$a(args, T)$$

where a is the operator, T the time argument of the action, and $args$ any other (ground) arguments for the action. Also, again as in D4, we will assume that actions are represented within KB_{plan}, KB_{react}, KB_{TR} as

$$assume\_happens(a(args), T)$$

Similarly for timed fluent literals.

Further, as in D4, we will give KB_0 as a set of (existentially quantified) facts of the form

$$executed(a(args, T), \tau)$$
$$observed(l(args, T), \tau)$$
$$observed(a(args, \tau 0), \tau)$$

where a is an action operator, l is a fluent literal, T is an existentially quantified variable, τ, $\tau 0$ are ground constants (the concrete time of execution or observation). However, as in the revised D4, we will assume that the capabilities of planning, reactivity and temporal reasoning (and thus goal decision that uses temporal reasoning), use the ground variant of KB_0, namely the set of facts of the form

$$executed(a(args, \tau))$$
$$observed(l(args, \tau))$$
$$observed(a(args, \tau 0), \tau)$$

Finally, we will assume that the behaviour of computees is given via operational traces, defined as in D4, where passive observations from the environemnt (via POI) are always treated as interrupts.

3.1 ADAPTABILITY

Computees must be able to adapt their operation to the changing environment.

Reactivity – A computee may adapt its reactive behaviour after the occurrence of certain changes in the environment. For example, it may react differently to the same request, depending on what resources it currently owns.

Example 3.1 *Suppose that Francisco's computee f requests from a 'San Vincenzo Station' computee (svs in short) the arrival time of a certain train, say TR123, from Rome. Suppose that svs refuses to give this information as it does not have it. Then further suppose that svs later receives information for the arrival time of the TR123 train from a 'Central Office' computee (co in short). If f now requests the arrival time of TR123, svs will accept the request and provide the information.*

Below, we will illustrate the state and behaviour of computee svs. We will use utterances $query\_ref$, $refuse$, $inform$ *inspired by the utterances used in some of the FIPA specifications for communicative acts ([FIP01a], 3.16 and [FIP01b]) (see also section 4.4.2)* [1]

- **State**
  - KB^{svs}_{react}:

$$observed(C, tell(C, svs, query\_ref(Q), D, T0), T), holds\_at(have\_info(Q, I), T)$$
$$\Rightarrow assume\_happens(tell(svs, C, inform(Q, I), D), T') \wedge T' > T$$
$$observed(C, tell(C, svs, query\_ref(Q), D, T0), T), holds\_at(no\_info(Q), T)$$
$$\Rightarrow assume\_happens(tell(svs, C, refuse(Q), D), T') \wedge T' > T$$

$$holds\_initially(no\_info(arrival\_time(tr123))$$
$$precondition(tell(svs, C, inform(Q, I), D), have\_info(Q, I))$$
$$initiates(tell(C, svs, inform(Q, I), D), T, have\_info(Q, I))$$
$$terminates(tell(C, svs, inform(Q, I), D), T, no\_info(Q))$$

$$executable(tell(svs, C, S, D)) \leftarrow C \neq svs$$
$$assume\_happens(A, T), not\, executable(A) \Rightarrow false$$

$$assume\_happens(tell(svs, C, inform(Q, I), D), T),$$
$$assume\_happens(tell(svs, C, refuse(Q), D), T') \Rightarrow false$$

KB^{svs}_0 is empty to start with.

The remaining components of KB^{svs} do not play any role in this example, and can be considered to be empty.

  - $Goals^{svs}$={}
  - $Plan^{svs}$ ={}

- **Behaviour**

[1] Although we use the same names of communicative acts as in the FIPA specification, we do not adopt here their semantic specification.

– Via POI at time 5 [2] svs records in its KB_0:

$$observed(f, tell(f, svs, query\_ref(arrival\_time(tr123)), d, 3), 5).$$

Namely, svs observes at time 5 that f has requested (within a dialogue labelled by the identifier d) at the earlier time 3 to be informed about the arrival time of train TR123.

– Via RE svs adds to its $Plan$ the (top-level reactive) action:

$$\langle tell(svs, f, refuse(arrival\_time(tr123)), d, T'), \bot, true, \{T' > 5\}\rangle.$$

This is generated via the reactive capability:

$KB \models_{react} assume\_happens(tell(svs, f, refuse(arrival\_time(tr123)), d), T'), T'>5$,

as svs does not have information about the requested arrival time information.

– Via AE, svs executes the action, say at time 10, thus adding to KB_0:

$executed(tell(svs, f, refuse(arrival\_time(tr123)), d, T'), 10)$.

– Now assume that at time, say, 17, via POI, svs records in its KB_0

$$observed(co, tell(co, svs, inform(arrival\_time(tr123), 18:00), d', 15), 17)$$

namely co gives svs the arrival time of train TR123, at time 15, without being prompted. [3]

– Also, at time, say, 25, again via POI, svs records in its KB_0

$$observed(f, tell(f, svs, query\_ref(arrival\_time(tr123)), d'', 20), 25)$$

namely f asks again for the arrival time of train TR123 within a new dialogue d'.

– Now, the reply of svs will be different. Via RE, say at time 28, svs adds to its $Plan$ the (top-level reactive) action

$\langle tell(svs, f, inform(arrival\_time(tr123), 18:00), d'', T''),$

$\bot,$

$have\_info(arrival\_time(tr123), 18:00), \{T'' > 28\}\rangle$

– Then, AE, say at time 30, generates

$$tell(svs, f, inform(arrival\_time(tr123), 18:00), d'', 30).$$

– Finally, PR clears the state.

Adjustment/Abandonment – If a perceived change in the environment renders a goal of a computee unachievable, then the computee will abandon that goal. Analogously, if a change in the environment renders (some actions in) a plan of the computee impossible to execute, then the computee will adjust (part of) its plan accordingly.

Example 3.2 *Consider the situation where Francisco, while staying in Rome for the weekend, asks his computee to buy him a ticket for the jazz festival that is taking place on Saturday evening. The computee finds out that the festival is sponsored by Alitalia and, as a result, a limited number of free tickets are reserved to Alitalia club members. As a result it plans to acquire a ticket for Francisco in this way. However, the computee is informed by the festival's*

[2] Here and in the rest of this section, the time of executed communicative action and of the observation could be different in a concrete implementation, as they depend on the time of execution of transitions within the operational trace of a computee.

[3] d' is the identifier of the dialogue within which this utterance has been performed, and would typically be different from the earlier d.

organisation that this type of tickets have all been acquired, so the computee suspends the planned acquiring action as undoable. Then the plan is adjusted by the computee now deciding to buy the ticket instead of acquiring it via the club membership route. However, when the computee finds out that the festival has been sold out, then it drops all the planned actions as undoable and the goal as unachievable.

- **State**

  - KB_{plan}:
    $initiates(acquire(O), T, have(O))$
    $precondition(acquire(festival\_ticket), member\_offer)$
    $initiates(buy(O), T, have(O))$
    $precondition(buy(O), \neg sold\_out(O))$
    $holds\_initially(member\_offer)$
    $holds\_initially(\neg(sold\_out(festival\_ticket)))$
    $initiates(tell(festival\_org, f, sold\_out, D), T, sold\_out(festival\_ticket))$
    $terminates(tell(festival\_org, f, no\_member\_seats, D), T, member\_offer)$
    $executable(acquire(O))$
    $executable(buy(O))$
    $assume\_happens(A, T), not\, executable(A) \Rightarrow false$

    KB_0 is empty to start with.

    The remaining components of KB do not play any role in this example, and can be considered to be empty.

  - $Goals$=$\{g_1 = \langle have(festival\_ticket, T), \perp, \{T \leq 10\}\rangle\}$
    where 10 stands for the time by when the ticket must be available to Francisco for him to attend the festival. Note that this goal may be reactive or not, as this distinction does not play any role in this example.
  - $Plan = \{\}$

- **Behaviour**

  - Via PI, say at time 1, f generates an action
    $a_1 = \langle acquire(festival\_ticket, T'), g_1, member\_offer(T'), \{1 < T' < T\}\rangle$
  - Via POI, say at time 6, the action by the festival organisation:
    $tell(festival\_org, f, no\_member\_seats, d, 5)$
    is recorded into KB_0 as
    $observed(festival\_org, tell(festival\_org, f, no\_member\_seats, d, 5), 6)$
  - The computee does not realise that the preconditions of a_1 is currently false, and would execute a_1 within AE, say at time 7.

- At time, say, 8, AOI is then executed [4] for the effect $have(festival\_ticket, T)$ of action a_1, recording into KB_0

  $observed(\neg have(festival\_ticket, \_), 8)$
- Now, PI is applied, say at time 9, and it introduces the new action

  $a_2 = \langle buy(festival\_ticket, T^*), g_1, \neg(sold\_out(festival\_ticket)), 9 < T^* < T \rangle$
- Via POI, say at time 10, the information

  $tell(festival\_org, f, sold\_out(festival\_ticket), d', 8)$

  is recorded into KB_0 as

  $observed(festival\_org, tell(festival\_org, f, sold\_out(festival\_ticket), d', 8), 10)$
- The GR transition, say at time 11, realises that goal g_1 is now timed out, and drops it from $Goals$.
- Then, the PR transition, say at time 12, abandons a_1 and a_2.

Suspension/Introduction –

Due to some change in the environment, a goal or action may be temporarily suspended (not selected) to give priority to a more preferred goal or action. In particular, a "more urgent" goal/action might be preferred over a "less urgent" goal/action. Also, a goal may be introduced in the sense of being brought into the foreground due to a change of preferences of the computee.

Example 3.3 *(Suspension and introduction of goals)*
In this example we examine how Francisco's computee can assist Francisco to leave San Vincenzo. To illustrate this we assume that Francisco's computee contains in its goal decision theory, KB_{GD}, a personality theory on needs and motivations of the kind discussed in [KM02] and re-interpreted in [WS03]. Following the re-interpretation proposed in [WS03] the computee has its possible goal decisions labelled (or separated) into five major categories based on the needs each goal decision contributes to fulfill: operational*, self-benefit, peer-interaction, community-interaction, and non-utilitarian (see [WS03] for an explanation).*

In our example, Francisco's computee, has to take into consideration, in addition to the goal (or need) of Francisco to leave San Vincenzo, the needs of Francisco as part of his peer-interaction motivations, for example, Francisco's everyday needs. These include context-independent and recurring needs, such as reading daily news, and context-dependent needs, such as leaving San Vincenzo at 14:00h. The computee also has to take into consideration its own needs, for example, its operational needs dictate to take appropriate action when Francisco's PDA is running out of battery power. Possible goal decisions contributing to the fulfillment of all these needs are encoded in Francisco's computee's knowledge base for goal decision, KB_{GD}.

In KB_{GD}, a low-battery alert from the PDA has higher priority over all other needs, such as aggregation of news feeds and context-dependent needs, like catching a train. Similarly, context-dependent needs are ranked higher than context-independent and recurring needs like reading of news. In other words, the context-dependent need of Francisco to catch the 14:00h train to leave San Vincenzo has higher priority than aggregating news feeds on the day of the departure.

[4] For this we assume that the auxiliary part of $\mathcal{T}_{cycle}$ contains the fact

$$unreliable\_post(S) \leftarrow \langle acquire(X, T), \_, \_ \rangle \in S$$

However, a low-battery alert from the PDA is always preferred in order to avoid loosing all the PDA data.

- **State**

  Initially, $Goals$, $Plans$, and KB_0 are empty. Moreover, KB_{TR} is as given in D4, with an empty domain-dependent part. Furthermore, except for KB_{GD} defined below, all other components of KB play no role in this example, and can be thought to be empty.

  – KB_{GD}^{low}: There are three possible goals: (a) Leaving San Vincenzo (lsv), (b) aggregation of news feeds (nfa), and (c) low-battery alert (lba). The knowledge base has the following rules:

$$
\begin{aligned}
gd(lsv) : \ & lsv(T), T < T' \leftarrow \\
& \quad holds\_at(finished\_work, T_{now}), \\
& \quad T' = T_{now} + 6. \\
gd(nfa) : \ & nfa(T), T < T' \leftarrow \\
& \quad holds\_at(away, T_{now}), \\
& \quad T' = T_{now} + 3. \\
gd(lba) : \ & lba(T), T < T' \leftarrow \\
& \quad holds\_at(low\_battery, T_{now}), \\
& \quad T' = T_{now} + 2.
\end{aligned}
$$

  – KB_{GD}^{aux}: Each goal is assigned a category, respectively, *required*, *optional*, and *operational*, and priority relationships amongst these categories are specified, i.e.:

$$
\begin{aligned}
& typeof(lsv, required) \\
& typeof(nfa, optional) \\
& typeof(lba, operational) \\
& more\_urgent\_wrt\_type(required, optional) \\
& more\_urgent\_wrt\_type(operational, required) \\
& more\_urgent\_wrt\_type(operational, optional)
\end{aligned}
$$

  Note also that the three goals are pairwise incompatible, i.e. that the computee can only do one of these goals at a time.

  – KB_{GD}^{high}: The following preferences apply:

$$
\begin{aligned}
gd\_pref(X, Y) : gd(X) < gd(Y) \quad \leftarrow \quad & typeof(X, XT), \\
& typeof(Y, YT), \\
& more\_urgent\_wrt\_type(YT, XT).
\end{aligned}
$$

- **Behaviour**

– Initially, at $T_{now} = 1$, a GI transition is executed. This calls the GD capability, giving

$$KB \models_{GD} \langle lsv(T), \{T < 7\}\rangle$$

Note that Francisco has finished his work and that he's currently away from home. Hence, the goal of leaving San Vincenzo (lsv) is preferred over the other goals since it contributes to a required need and there are no operational needs, such as a low-battery alert, since there has been no low-battery signal observed so far.

Thus, executing the GI transition gives the goal of Leaving San Vincenzo (lsv) before time 7, which is thus introduced within the (non-reactive goals of the) state of Francisco's computee f, i.e.:

$$Goals = \{g_1 = \langle lsv(T), \bot, \{T < 7\}\rangle\}$$

– Next, we suppose that a POI occurs at $T_{now} = 2$ which adds an observation about a low-battery signal from Francisco's PDA,i.e.:

$$KB_0 = \{observed(low\_battery(T'), 2)\}$$

– Then, Francisco's computee, through the interrupt part of its cycle theory chooses at $T_{now} = 3$ again a GI transition. Now, because of the extra information acquired, the GD capability returns a low-battery alert (lba) as its top-level (non-reactive) goal and "suspends" (by temporarily dropping) the goal of leaving San Vincenzo. Hence, $Goals$ becomes:

$$Goals = \{g_2 = \langle lba(T''), \bot, \{T'' < 5\}\rangle\}$$

Note that we can have an alternative KB_{GD} theory that renders the two goals lsv and lba equally preferred, and hence after the observation of $low\_battery$ the $Goals$ will contain both g_1 and g_2 as its top-level (non-reactive) goals. Depending on the the profile of the computee, as expressed by its cycle theory, the cycle theory might impose as the next transition a PI transition that is required to plan for one or the other goal or both. For example, a *punctual* computee will choose to plan for the lba goal as it is (more) urgent then the lsv goal.

Example 3.4 *(Introduction of actions)*
In this example, while trying to achieve the goal of leaving San Vincenzo, Francisco's computee receives a request (by means of a POI transition) from another computee to share a taxi. Francisco's computee first introduces the action to accept the request, as it believes that Francisco is out of cash and could not afford a taxi alone. (The computee also believes that Francisco would want to share a taxi if he knew that there was one single taxi available, but this rule does not play any role in the example below, as Francisco's computee that there are many taxis available.) Then, after realising that instead Francisco has sufficient cash for the taxi, it drops the previously introduced action and introduces a new action, to reject the request instead.

- **State**
  KB_{react} of f is given by:

$$holds\_at(want(taxi), T),$$
$$observed(C, tell(C, f, request(share\_taxi), D, T0), T),\ holds\_at(\neg have(cash), T)$$
$$\Rightarrow assume\_happens(tell(f, C, accept(share\_taxi), D), T') \wedge T' > T$$

$$holds\_at(want(taxi), T),$$
$$observed(C, tell(C, f, request(share\_taxi), D, T0), T), holds\_at(single\_taxi, T)$$
$$\Rightarrow assume\_happens(tell(f, C, accept(share\_taxi), D), T') \wedge T' > T$$

$$holds\_at(want(taxi), T),\ holds\_at(have(cash), T),$$
$$observed(C, tell(C, f, request(share\_taxi), D, T0), T),\ holds\_at(\neg single\_taxi, T)$$
$$\Rightarrow assume\_happens(tell(f, C, reject(share\_taxi), D), T') \wedge T' > T$$

$$assume\_happens(tell(f, C, accept(share\_taxi), D), T),$$
$$assume\_happens(tell(f, C, reject(share\_taxi), D), T') \Rightarrow false$$

$$holds\_initially(\neg have(cash))$$
$$holds\_initially(\neg single\_taxi)$$
$$terminates(take(taxi), T, want(taxi))$$
$$precondition(tell(f, C, accept(share\_taxi), D), want(taxi))$$

KB_0 is initially empty.

The remaining components of KB do not play any role in this example, and can be assumed to be empty.

- $Goals = \{g_1 = \langle want(taxi, T), \perp, \{T < 10\}\rangle\}$ (g_1 could be reactive or not).
- $Plan$ is initially empty.

Behaviour

- Via POI, say at time 6, f records that another computee (simply o) has requested f (at an earlier time) to share a taxi, e.g. KB_0 is updated to contain:
  $observed(o, tell(o, f, request(share\_taxi), d, 4), 6)$
- Then, via RE, say at time 7, f adds to its $Plan$ the (reactive) action
  $a_1 = \langle tell(f, C, accept(share\_taxi), d, T'), \perp, want(taxi), \{T' > 7\}\rangle$
- Alternatively, suppose that f passively observes, via POI at time, say, 3, that, contrary to its current belief, Francisco does have cash, namely the following fact is added to KB_0
  $observed(have(cash, \_), 3)$
- In this case, via RE, at time 7, f introduces into $Plan$ the (reactive) action
  $a_2 = \langle tell(f, C, reject(share\_taxi), d, T''), \perp, true, \{T'' > 7\}\rangle$

3.2 TOLERANCE TO PARTIAL INFORMATION

The computee should continue to operate and take decisions despite the incompleteness of the available information and the increase of information available to it over time.

Conditional Decisions – A computee should be able to set itself a plan conditional on information about the environment that is not available to the computee at the time of decision of the plan.

Example 3.5 *In preparation of Francisco's next business trip, Francisco's computee aims at getting a plane ticket from Madrid to Denver. One possible way to buy plane tickets is over the Internet. Buying tickets this way is usually possible, but not to all destinations (depending on whether the airlines flying to the destinations do sell tickets over the Internet) and not without an Internet connection. The computee does not currently have the connection, nor the information that Denver is indeed a destination for which tickets can be bought on line. It plans to buy the ticket over the Internet nonetheless, conditionally, but checks the conditions before executing the action.*

- **State**
  - KB does not play any role in this example, except for generating the action a_1 below in $Plan$ and goals g_2 and g_3, which we assume has been done beforehand. [5] So we will ignore KB.
  - $Goals = \{g_1 = \langle have\_ticket(T'), \perp, \{T' < 15\}\rangle,$
    $g_2 = \langle available\_connection(T''), g_1, \{T'' = T\}\rangle,$
    $g_3 = \langle available\_destination(T'''), g_1, \{T''' = T\}\rangle\}$
  - $Plan = \{a_1 = \langle buy\_ticket(T), g_1, available\_connection(T) \wedge$
    $available\_destination(T), \{T < T'\}\rangle\}$ [6]
- **Behaviour**
  - Via SI, the following actions are added to $Plan$ [7]
    $a_2 = \langle sense\_precondition(available\_connection(T_1)), g_1, true, \{T_1 < T\}\rangle$
    $a_3 = \langle sense\_precondition(available\_destination(T_2)), g_1, true, \{T_2 < T\}\rangle$
  - Then, via AE, these actions are executed (before the original action), and KB_0 is changed accordingly. E.g., if $\models^{\tau}_{Env}$ $available\_connection(\tau)$, say for $\tau = 5$, then the following fact is added to KB_0 after AE at time 5:
    $observed(available\_connection(T_1), 5)$
    If $\models^{\tau}_{Env}$ $\neg available\_connection(\tau)$, again for $\tau = 5$, then the following fact is added to KB_0 after AE at time 5:

[5] Note that every plan that is generated must satisfy the integrity constraints in KB_{plan}. In particular, any precondition of actions in plan that do not already hold must be generated as sub-goals in $Goals$. This is why g_2 and g_3 occur in $Goals$ below. Henceforth we will sometimes ignore these additional subgoals for the sake of simplicity of presentation.

[6] g_1 and a_1 can be reactive or not, as this does not matter for this example.

[7] For this we assume that the auxiliary part of $\mathcal{T}_{cycle}$ contains the fact

$$unreliable\_pre(S) \leftarrow \langle available\_connection(\_) \in S$$

$observed(\neg available\_connection(T_1), 5)$

Finally, if neither

$\models^{\tau}_{Env}$ $available\_connection(\tau)$

nor

$\models^{\tau}_{Env}$ $\neg available\_connection(\tau)$,

again for $\tau = 5$, then no fact is added to KB_0. Similarly for the other precondition. Let us assume that after this step of AE, KB_0 becomes

$observed(available\_connection(T_1), 5)$

$observed(available\_destination(T_2), 5)$

– AE can then execute the original action a_1. Note that the computee might decide to execute the action even if one or both preconditions are not known to be satisfied after the sensing.

– Then, via GR, g_1 is dropped.

– Finally, via PR, a_1, a_2, a_3 are dropped.

Adjustment – The goals and plans of a computee are adjusted when this acquires new knowledge from the environment that was not available to the computee at the time of deciding these goals and plans.

Example 3.6 *Francisco's computee has the goal to have a museum ticket for some (state-run) museum that Francisco wants to visit, and a plan to buy it. The computee, before even executing the plan, observes that it is the European Heritage day (via an appropriate "message" from the computee eu, the European Union) and all state-run museums in Europe are free on that day. Then, its goal is already achieved and both goal and plan are deleted.*

- **State**

  – KB_{TR} contains

  $initiates(ehd, T, have(ticket))$

  $initiates(buy(O), T, have(O))$

  $precondition(buy(O), have\_money)$

  The remaining components of KB do not play any role in this example, and can be considered to be empty.

  – $Goals = \{g_1 = \langle have(ticket, T), \bot, \{T \leq 10\}\rangle,$
  $g_2 = \langle have\_money(T''), g_1, \{T'' = T'\}\rangle\}$

  – $Plan = \{a_1 = \langle buy(ticket, T'), g_1, have\_money(T'), \{T' < T\}\rangle\}$ [8]

- **Behaviour**

  – Via POI, e.g. at time 6, the following fact is added to KB_0:

  $observed(eu, ehd(Time), 6)$

  – Via GR, e.g. at time 8, g_1 and g_2 are eliminated from $Goals$, as g_1 is already achieved.

  – Via PR, e.g at time 10, a_1 is eliminated from $Plan$, as g_1 is not in $Goals$ anymore.

[8] g_1 and a_1 can be reactive or not, as this does not matter for this example.

3.3 OPENNESS

Societies are dynamic, in that computees might join and leave them at all times. As a result of this dynamicity, the computees already within the dynamically changing societies need to adjust their decisions to reflect the new state of the environment, due to the addition/deletion of other computees (**OPENNESS (1)**). Moreover, the computees that move, thus joining new societies, should adjust their decisions and behaviour to take into account the different expertise and the different social rules available in the new society (**OPENNESS (2)**).

3.3.1 OPENNESS (1)

Adjustement – The computee should adjust its decisions on plans and goals to pursue when it becomes aware of new computees in the societies to which it belongs. These new computees might provide new information that was not available earlier. The adjustment might render the new plans and goals easier to achieve.

Example 3.7 *Francisco's computee tries to get a resource which it needs for a particular plan to achieve one of its goals, for example, find a person that is likely to accept to share a taxi (artst* – accepted request to share a taxi*) with Francisco to the train station in order to* reach San Vincenzo's train station *(rsvts). Francisco's computee has asked all other available computees in the hotel society for the resource, but did not manage to get it, so it suspends its goal. But, when a new computee becomes available in the hotel, then Francisco's computee can go back and ask the new computee.*

- **State**
  - KB^{f}_{plan} consists of the following domain-dependent knowledge, inspired by [STT02]:

$$holds\_at(reach\_station, T) \leftarrow holds\_at(accepted\_share, T'), T' < T$$

$$\begin{aligned} holds\_at(accepted\_share, T) \quad \leftarrow \quad & holds\_at(customer(O), T'), \\ & happens(tell(f, O, request(share\_taxi), d(ex7, T')), T'), \\ & holds\_at(accepted\_request(O, share\_taxi), T''), \\ & T' < T'' < T \end{aligned}$$

$$\begin{aligned} & precondition(tell(f, O, request(share\_taxi), D), \neg already\_rejected(O)) \\ & initiates(tell(O, f, reject(request(share\_taxi)), D), \\ & \qquad T, \\ & \qquad already\_rejected(O)) \end{aligned}$$

$$\begin{aligned} & initiates(tell(O, f, accept(request(share\_taxi)), D), \\ & \qquad T, \\ & \qquad accepted\_request(O, share\_taxi)) \end{aligned}$$

These definitions basically impose to ask one after the other, in sequence, computees of customers of the hotel, without asking the same computee twice, until one accepts the offer.

Knowledge about customers in the hotel can be drawn from the following domain-dependent definitions, inspired by [TS02]:

$holds\_initially(customer(o1))$
$holds\_initially(customer(o2))$
$holds\_initially(\neg already\_rejected(o1))$
$holds\_initially(\neg already\_rejected(o2))$
$initiates(join(C), T, customer(C))$

Francisco's computee cannot join societies on someone else's behalf, namely, it can only execute the action of joining itself:

$executable(join(f), T))$
$executable(tell(f, C, S, D)) \leftarrow C \neq f$
$assume\_happens(A, T), not\, executable(A) \Rightarrow false$

KB_0 is initially empty.

The other components of KB do not play any role in this example, and can be considered to be empty.

- $Goals = \{g_1 = \langle reach\_station(T), \perp, \{T < 20\}\rangle$,
- $Plan = \{\}$

- **Behaviour**
  - Via PI, the following action a_{o1} and goal g_{o1} are added to $Plan$ and $Goals$, respectively: [9]
    $\langle tell(f, o1, request(share\_taxi), d(ex7, T'), T'), g_1, \neg already\_rejected(o1), \{T'<T''\}\rangle$
    $\langle accepted\_request(o1, share\_taxi, T''), g_1, \{T'' < T\}\rangle$
  - Via AE, a_{o1} is then executed, say at time 1, thus adding to KB_0
    $executed(tell(f, o_1, request(share\_taxi), d(ex7, T'), T'), 1)$
  - Via POI, the following observation is then added to KB_0, say at time 3:
    $observed(o_1, tell(o_1, f, reject(request(share\_taxi)), d(ex7, T'), 3), 3)$
  - Via GR and PR, g_{o1} and a_{o1} are then deleted from $Goals$ and $Plan$, respectively.

[9] Note that in the abstract model for computees in D4, as well as in the computational model for computees in D8, all actions and goals in the state of a computee are ground except for the time parameter. Here, instead, the dialogue identifier $d(ex7, T')$ is not ground. However, as we will see, it will be instantiated when the action a_{o1} is executed.

– Via PI, the following action a_{o2} and goal g_{o2} are added to *Plan* and *Goals*, respectively:

  $\langle tell(f,o2,request(share\_taxi),d(ex7,T^*),T^*),g_1,\neg already\_rejected(o1),\{T^*<T^{**}\}\rangle$
  $\langle accepted\_request(o2, share\_taxi), T^{**}), g_1, \{T^{**} < T\}\rangle$

– Via AE, a_{o2} is then executed, say at time 5, thus adding to KB_0
  $executed(tell(f, o_2, request(share\_taxi),d(ex7, T^*), T^*), 5)$

– Via POI, the following observation is then added to KB_0, say at time 6:
  $observed(o_2, tell(o_2, f, reject(request(share\_taxi)), d(ex7, T^*), 6), 6)$

– Via GR and PR, g_{o2} and a_{o2} are then deleted from *Goals* and *Plan*, respectively.
  At this point there is nothing more the computee can do to achieve its goal g_1. However, g_1's deadline is not approaching, and f may decide to suspend g_1.

– Suppose that, soon after this, f observes, via POI, that a new computee has joined the hotel, namely KB_0 is extended with
  $observed(o3, join(o3, 3), 10)$

– Then f, via PI, adds the following action a_{o3} and goal g_{o3} to its *Plan* and *Goals*, respectively:

  $\langle tell(f,o3,request(share\_taxi),d(ex7,T^!),T^!),g_1,\neg already\_rejected(o3),\{T^! < T^{!!}\}\rangle$
  $\langle accepted\_request(o3, share\_taxi), T^{!!}), g_1, \{T^{!!} < T\}\rangle$

– Via AE, a_{o3} is then executed, say at time 12, thus adding to KB_0
  $executed(tell(f,o3,request(share\_taxi),d(ex7, T^!), T^!), 12)$

– Via POI, the following observation is then added to KB_0, say at time 15:
  $observed(o_3, tell(o_3, f, accept(request(share\_taxi)), d(ex7, T^!), 15), 15)$

– Via GR and PR, g_{o3}, g_1 and a_{o3} are then deleted from *Goals* and *Plan*, respectively.
  The presence of the new computee has allowed f to achieve its goal successfully.

Conformance – A particular change in the configuration of the environment of a computee is a change in the social rules of the society to which it belongs. In such a case the computee should change its goals and plans to conform to these new rules.

Example 3.8 *Suppose that a hotel reception computee r (representing the receptionist), controls the resource of checking out customers (via their computees) within a hotel society. Another computee d is deputy of the first in controlling that resource (this computee too represents a real person that deputises the receptionist when necessary). The computees have the following knowledge: if somebody asks to check out then the receptionist should reply but if the receptionist has left the society (hotel), then the deputy should reply. Assume the receptionist leaves. Then the behaviour of the deputy changes as prescribed.*

• **State**

  – Within KB of r we have the following components, inspired by [TS02]:

  KB^r_{react}:
  $observed(C, tell(C, h\_s, request(check\_out), D, T0), T),$

$$holds\_at(in(r, h\_s), T) \Rightarrow assume\_happens(tell(r, C, ok(check\_out), D, T')) \wedge T' > T$$

$holds\_initially(in(r, h\_s))$
$terminates(leave(X, h\_s), T, in(X, h\_s))$
$initiates(enter(X, h\_s), T, in(X, h\_s)) \leftarrow holds\_at(member(X, h\_s), T)$
$holds\_initially(member(r, h\_s))$

$executable(leave(r))$
$executable(enter(r, h\_s))$
$executable(tell(r, C, S, D)) \leftarrow C \neq r$
$assume\_happens(A, T), not\, executable(A) \Rightarrow false$

Intuitively, r knows that it should reply to any check out request to the hotel society (h_s) whenever it is in the hotel. Note that here we are assuming, as in [TS02], that communication utterances can be addressed to a society rather than to an individual computee. This is useful when, as in this example, different members of the society might have to deal with the utterance at different times, and the utterer does not know or is not allowed to know which member is responsible when.
In this example, the computee r also knows that it is in the hotel to start with, that leaving the hotel terminates being there, that it is a member of the hotel staff, and that it (and any other computee) can always come back into the society simply by (re-)entering it.
Finally, the computee knows that it can only execute some actions.

KB_0 of r is empty to start with.

The remaining components of KB of r do not play any role in this example, and can be considered to be empty.

– Within KB of d we have the following components, again inspired by [TS02]::

KB^d_{react}:
$observed(C, tell(C, h\_s, request(check\_out), D, T0), T),$
$holds\_at(\neg in(r, h\_s), T),$
$holds\_at(in(d, h\_s), T),$
$$\Rightarrow assume\_happens(tell(d, C, ok(check\_out), D), T') \wedge T' > T$$

$holds\_initially(in(r, h\_s))$
$holds\_initially(in(d, h\_s))$
$terminates(leave(X, h\_s), T, in(X, h\_s))$
$initiates(enter(X, h\_s), T, in(X, h\_s)) \leftarrow holds\_at(member(X, h\_s), T)$
$holds\_initially(member(d, h\_s))$
$holds\_initially(member(r, h\_s))$

$executable(leave(d))$
$executable(enter(d, h\_s))$
$executable(tell(d, C, S, D)) \leftarrow C \neq d$
$assume\_happens(A, T), not\, executable(A) \Rightarrow false$

Intuitively, d knows that it should reply to any check out request to the hotel society (h_s) whenever it is in the hotel and the receptionist is not. It also knows that leaving the hotel terminates being there, that it and the receptionist are in the hotel to start with, that it and the receptionist are members of the hotel staff, and that any computee that is a member of staff in the hotel can always come back into the hotel society simply by (re-)entering it. Finally, the computee knows that it can only execute some actions.

KB_0 of d is empty to start with.

The remaining components of KB of d do not play any role in this example, and can be considered to be empty.

– $Goals$ and $Plan$ of both r and d are initially empty.

- **Behaviour**
  - Via POI, say at time 5, a new check-out request from a customer, say c_1, is recorded into the KB_0 of d as
    $observed(c_1, tell(c_1, h\_s, request(check\_out), di, 1), 5)$
  - The same request, but, say, at time 7, is recorded into the KB_0 of r as
    $observed(c_1, tell(c_1, h\_s, request(check\_out), di, 1), 7)$
  - Via RE, r adds to its $Plan$ the action
    $a = \langle tell(r, c_1, ok(check\_out)), di, T), \bot, true, \{7 < T\}\rangle$
  - Via AE, r executes the action a at time, say, 10, thus changing KB_0 accordingly.
  - In the meanwhile, the state of d is unchanged (except for the change to KB_0). Then, at time, say, 12, r leaves the hotel temporarily, and, via POI, d observes it at time, say, 15, recording into its KB_0:
    $observed(r, leave(r, h\_s, 12), 15)$
  - Assume now that a new check-out request arrives, say from customer c_2, and this is recorded, via POI, say at time 27, into the KB_0 of d as
    $observed(c_2, tell(c_2, h\_s, request(check\_out)), dj, 21), 27)$
  - Then, via RE, say at time 28, d adds to its $Plan$ the action
    $b = \langle tell(d, c_2, ok(check\_out)), dj, T'), \bot, true, \{28 < T'\}\rangle$
  - Via AE, d executes the action b at time, say, 30, thus changing KB_0 accordingly.

Variants of this example are possible, e.g. one in which the receptionist actually notifies the deputy verbally that it is leaving, so as to achieve increased coordination.

3.3.2 OPENNESS (2)

Exploitation – A computee should be able to adapt its current decisions on goals and plans when it moves to a new society to exploit the characteristics of the new society.

Example 3.9 *A computee moves to a society where a certain resource is freely available. If it had plans to get that resource, it can now simplify the plan. This example manifests itself in the situation where Francisco's computee wants to access information about trains while situated in the hotel and taxi (managed by their corresponding societies). In this case, in order for the computee to receive train information Francisco's mobile phone account needs to be charged for the service to be provided, and Francisco's computee plans to do this. However, before the computee even has time to execute this plan, Francisco arrives at the station. Here, information about trains is freely available, and Francisco's computee can drop the plan to charge the phone.*

- **State**
  - KB_{plan} and KB_{TR} contain
    $initiates(buy(O), T, have(O))$
    $holds\_at(have(train\_info), T) \leftarrow holds\_at(joined(station), T)$
    $holds\_at(at(station), T) \leftarrow holds\_at(joined(station), T)$
  - $Goals = \{g_1 = \langle at(station, T1), \bot, \{T1 \leq 10\}\rangle,$
    $g_2 = \langle have(train\_info, T), \bot, \{T \leq T1 + 2\}\rangle\}$
  - $Plan = \{a_1 = \langle take(taxi, T0), g_1, true, \{T0 < T1 - 5\}\rangle,$
    $a_2 = \langle buy(train\_info, T'), g_2, true, \{T' < T\}\rangle\}$
- **Behaviour**
  - Via AE, f executes the action of taking the taxi at time 5, recording into its KB_0:
    $executed(take(taxi, T0), 5)$
  - Via POI, f observes and records into its KB_0 that it has joined the station at time 6:
    $observed(joined(station, T''), 6)$
  - Via GR, g_1 and g_2 are dropped from $Goals$.
  - Then, via PR, a_1 and a_2 are also dropped.

Adjustment – A computee should be able to adjust its current decisions on goals and plans when it leaves a society and thus some of these decisions become inappropriate or invalid.

Example 3.10 *A computee, e.g. Francisco's, should stop paying the subscription fee to a society once it has left that society.*

- **State**
  KB^f is as follows, inpired by [TS02]:
  - KB^f_{react} contains
    $holds\_initially(member(f, socs))$
    $terminates(tell(f, C, leave(S), D, T), member(f, S)) \leftarrow auth(C, S)$

$auth(john, socs)$
$observed(first\_of\_month, T), holds\_at(member(f, S), T) \Rightarrow$
$\qquad assume\_happens(pay(S), T') \wedge T \leq T' \leq T + 5$

KB_0 is initially empty.

- $Goals = \{\}$
- $Plan = \{\langle tell(f, john, leave(socs), d, T), \perp, true, \{T < 20\}\}$ (we can assume this action to be reactive).

- **Behaviour**
  - Via POI at time 1, f adds to KB_0 the fact
    $observed(first\_of\_month, 1)$
  - Via RE at time 2, f adds to $Plan$ the (reactive) action
    $\langle pay(socs, T'), \perp, \{\}, 1 \leq T' \leq 6\rangle$
  - Via AE at time 3, f adds to KB_0 the fact
    $executed(pay(socs, T'), 3)$
  - Now, via AE at time 10, f adds to KB_0 the fact
    $executed(tell(f, john, leave(socs), d, T), 10)$
  - Then, via POI, f adds to KB_0 the fact
    $observed(first\_of\_month, 20)$
  - Now, via RE, the reactive part of $Plan$ becomes empty.

3.4 HETEROGENEITY

The model should support a variety of different types of heterogeneous computees. Societies should be tolerant of this heterogeneity.

Expertise – A modular change on the internal knowledge of the computee or on its capabilities should be reflected directly on the decisions of the computee under otherwise exactly the same external environment and circumstances.

Example 3.11 *Francisco's computee's KB_{react} contains a rule saying to provide truthful answers to every request of information about Francisco by other computees or people, no matter how private. Then the computee's KB_{react} is changed so that that rule is replaced by another saying to give no information away, when requested, no matter how harmless. The reply to the same request will be different in the two cases.*

- **State 1**
  - KB_{react} contains the rule
    $observed(C, tell(C, f, request(give(X)), D, T), T0) \Rightarrow$
    $\qquad assume\_happens(tell(f, C, accept(give(X)), D), T') \wedge T' > T0$

  $executable(tell(f, C, S, D)) \leftarrow C \neq f$
  $assume\_happens(A, T), not\, executable(A) \Rightarrow false$

- $Goals = \{\}$
- $Plan = \{\}$

- **State 2**
  - KB_{react} contains the rule
    $observed(C, tell(C, f, request(give(X)), D, T), T0) \Rightarrow$
    $assume\_happens(tell(f, C, refuse(give(X)), D), T') \wedge T' > T0$
    $executable(tell(f, C, S, D)) \leftarrow C \neq f$
    $assume\_happens(A, T), not\, executable(A) \Rightarrow false$
  - $Goals = \{\}$
  - $Plan = \{\}$

- **Behaviour**

  Given State 1,

  - Via POI at time 5 the computee f adds to its KB_0 the fact
    $observed(john, tell(john, f, request(phone\_number), d, 3), 5)$
  - Via RE, the computee adds to its $Plan$ the action
    $\langle tell(f, john, accept(give(phone\_number), d, T'), \bot, \{\}, T' > 5\rangle$
  - Via AE at time, say, 10, the computee adds to its KB_0 the fact
    $executed(tell(f, john, accept(phone\_number), d, T'), 10)$

  Given State 2 instead,

  - Via POI the computee records thee same fact as for State 1 (assuming the execution times of the transitions are the same).
  - Via RE, the computee adds to its $Plan$ the action
    $\langle tell(f, john, refuse(phone\_number), d, T'), \bot, \{\}, T' > 5\rangle$
  - Then, the outcome of AE will also be different.

Overall Behaviour – The overall behaviour of a computee should be identifiable in terms of pre-defined categories. These categories should be linked to the method of internal operation of the computee and the way that this is parameterised. A modular component of the model should capture this method and parameters. A computee should exhibit different decisions and behaviour when we change, in a modular way, this component of internal operation of the computee.

Example 3.12 *(Heterogeneous operational behaviour using different behaviour profiles)*
We consider the following situation: Francisco's computee has the goal of leaving San Vincenzo and the plan to initiate the checkout process, make the payment, and arrange transportation from the hotel to the train station. The computee asks for a taxi but it then realises that the action failed by receiving some appropriate observation that has evidence for this.

This example aims to show that a computee would exhibit different decisions for what to do next based on the assigned profile of behaviour as specified in deliverable D4. In this example, appropriate variants of the normal cycle theory *are presented and used.*

- **State** Let us suppose the computee is currently in the following state (which is not an initial state):

  - $Goals = \{g_1, g_2\}$, where:

$$\begin{aligned} g_1 &= \langle lsv(T'), \bot, \{T' < 7\}\rangle \\ g_2 &= \langle get\_taxi(T), g_1, \{T \leq T'\}\rangle \end{aligned}$$

  - $Plans$=$\{a_1, a_2, a_3\}$, where:

$$\begin{aligned} a_1 &= \langle call\_taxi(T_1), g_2, \{T_1 = 3\}\rangle \\ a_2 &= \langle request\_porter(T_2), g_1, \{T_2 > T_1\}\rangle \\ a_3 &= \langle make\_payment(T_3), g_1, \{T_3 > T_2, T_3 < T\}\rangle \end{aligned}$$

  - KB_0 consists of:

$$\begin{array}{l} executed(call\_taxi(T_1), 3) \\ observed(\neg get\_taxi(T), 4) \end{array}$$

- **Behaviour** Below we show how each different type of computee will continue from this point according to its behaviour profile that regulates its operation. Specifically, what is the next transition decided by the cycle theory to be executed at time 6. We show the main preference rules of the cycle theory in each profile case.

  - A *punctual* computee would choose as its next cycle-step an AE transition to request a porter since action a_2 is qualified as urgent by the heuristic selection function h^u_{AS}, i.e. its deadline is about to expire, and the punctual profile dictates that the execution of an urgent action has higher priority than any other transitions. [10]

$$\begin{array}{l} punctual\_\_R^*_{AE|*}(S, As): \\ \qquad r_{*|AE}(S, As) > r_{*|*}(S, \_) \leftarrow h^u_{AS}(As). \end{array}$$

  - A *careful* computee would choose as its next step a PR transition prefering to revise its plan in the next cycle-step. The heuristic selection function h_{fail} is a non-empty set due to the fact that asking for a taxi in the previous transition failed (see evidence in KB_0). Hence, according to the careful profile of behaviour, a PR transition is preferred over other transitions.

$$\begin{array}{l} careful\_R^*_{PR|*}(S): \\ \qquad r_{*|PR}(S, \_) > r_{*|*}(S, \_) \leftarrow h_{fail}(As), As \neq \{\}. \\ careful\_R^*_{PR|*}(S): \\ \qquad r_{*|PR}(S, \_) > r_{*|*}(S, \_) \leftarrow h_{timeout}(As), As \neq \{\}. \end{array}$$

[10] Note here that, with respect to the notation in D4, we indicate in the name of rules in $\mathcal{T}_{behaviour}$ the pattern that the rules refer to, to increase readibility.

- A *focused* computee would choose as its next step an AE transition for action to request a porter (as in the case of the panctual computee) but for a different reason. It prefers to execute an action (an AE transition) in the next cycle-step from the same plan as the action that it has already executed. The heuristic selection function h_{sp} selects such actions if they exist. Hence, according to the focused pattern of behaviour, an AE transition is preferred over other transitions. This is expressed by the rule:

$$\begin{aligned}
&focused_R^{AE}_{AE|*}(S, As):\\
&\qquad r_{AE|AE}(S, As) > r_{AE|*}(S, _) \leftarrow\\
&\qquad\qquad last_executed_action(S, Last_A),\\
&\qquad\qquad h_{sp}(Last_A, As).
\end{aligned}$$

- An *impatient* computee prefers to suspend its current plan since it is invalidated by the environment (i.e. asking for a taxi failed) by preferring to execute other plans for other goals if these exist:

$$\begin{aligned}
&impatient_R^{*}_{AE|*}(S, As):\\
&\qquad r_{*|AE}(S, As) > r_{*|AE}(S, As') \leftarrow\\
&\qquad\qquad h_{fail}(Failed_As),\\
&\qquad\qquad h_{sp}(Failed_As, As'),\\
&\qquad\qquad \neg h_{sp}(Failed_As, As).
\end{aligned}$$

  As there are is only one goal currently in the state it will prefer to do a Goal Revision transition in the next step.
- A *cautious* computee chooses to do an AE transition next but for the action to making the payment rather than request for a porter. Asking for a porter is not qualified since that action's preconditions currently, namely that taxi is available at time 6 (computed via means of h^{pre}_{AS}) is not satisfied, since asking for a taxi did not succeed in the previous step.

$$\begin{aligned}
&cautious_R^{*}_{i|AE}(S, X):\\
&\qquad r_{*|i}(S, X) > r_{*|AE}(S, As) \leftarrow\\
&\qquad\qquad h^{pre}_{AS}(As, Pre),\\
&\qquad\qquad unsatisfied(Pre).
\end{aligned}$$

  where $unsatisfied(Pre)$ is evaluated using the temporal reasoning capability.
- Clearly, a computee that is both *punctual and careful* can choose either an AE or PR transition. However, by embedding a higher-order priority in our cycle theory, we ranked the careful profile as more important in the case of a timed out or failed action. Therefore, a PR transition is selected for the next cycle step of a computee assigned with this *hybrid profile* of behaviour.

$$\begin{aligned}
&punctual_and_careful_pref(Pref_1, Pref_2):\\
&\qquad Pref_1 > Pref_2 \leftarrow\\
&\qquad\qquad Pref_1 = careful_R^{*}_{*|*}(S),\\
&\qquad\qquad Pref_2 = punctual_R^{*}_{AE|*}(S, As).
\end{aligned}$$

4 Examples for WP2: societies of computees

In this section, we describe some simple examples designed to demonstrate the achievement of the (success) criteria for Workpackage 2 described at pages 18 through 21 of deliverable [LMM$^+$03], namely:

1. DEALING WITH PARTIAL INFORMATION
   (a) Social Expectations
   (b) Adjustment
2. OPENNESS
   (a) Semi-open society
3. ADAPTABILITY
   (a) Reactivity
   (b) Adjustment
4. HETEROGENEITY
   (a) Heterogeneity (1)
   (b) Heterogeneity (2)

4.1 DEALING WITH PARTIAL INFORMATION

The model should allow societies to be tolerant to partial information, by continuing to operate despite the incompleteness of the available information. Also, the model should cope with the increase of information on computees interactions available over time.

4.1.1 Social Expectations

At each time, the society has only a partial view of the history of events, since computees are completely autonomous and their future behaviour is not predictable. However, the society is able to devise a course of action (in the form of social expectations) reflecting the desired behaviour of computees, i.e., a behaviour that is compliant to interaction protocols and lets the society achieve its goal.

Let us consider the *Query* Protocol used between two computees A and B. This protocol is inspired by the FIPA specifications for communicative acts ([FIP01a], 3.16 and [FIP01b]),[11] and it works as follows: an agent A sends a query to an agent B. The query can either be a *query-if*, asking for an if/no answer, or a *query-ref*, asking for a referring expression. The answer could be *not-understood*, *refuse*, *failure*, or *inform*. A *refuse* answer indicates that B refuses to send any information, while *inform* is used to transmit the requested information.

In our case, we implement a simplified version of this protocol, which works as follows: a computee A requesting some information will tell *query-ref(info)* to a computee B during a conversation D, B is expected to answer back either *inform(info,answer)* or *refuse(info)*,

[11] As in section 3.4, although we use the same names of communicative acts as in the FIPA specification, we do not adopt here their semantic specification.

but not both of them. In the first case, *answer* will contain the requested information. We can easily imagine a situation where such interaction can occur in the San Vincenzo scenario, where Francisco's computee (computee A) asks a train station computee (computee B) for train information (see also section 3.4, example 3.1).
The protocol is defined below:

IC_1:

$$H(tell(A, B, query\text{-}ref(Info), D), T) \Rightarrow$$
$$E(tell(B, A, inform(Info, Answer), D), T_1) : T_1 < T + 10 \quad \vee$$
$$E(tell(B, A, refuse(Info), D), T_1) : T_1 < T + 10$$

IC_2:

$$H(tell(A, B, inform(Info, Answer), D), T) \Rightarrow$$
$$EN(tell(A, B, refuse(Info), D), T_1) : T_1 > T$$

IC_3:

$$H(tell(A, B, refuse(Info), D), T) \Rightarrow$$
$$EN(tell(A, B, inform(Info, Answer), D), T_1) : T_1 > T$$

Example 4.1 *Let the following history be given:* [12]

HAP:
{
$$H(tell(f, svs, query\text{-}ref(train\_info), d_1), 1).$$
}

Because of IC_1, the society should generate the following set of expectations:

EXP:
{
$$E(tell(svs, f, inform(train\_info, Info), d_1), T_1) : T_1 < 11 \quad \vee$$
$$E(tell(svs, f, refuse(train\_info), d_1), T_1) : T_1 < 11$$
}

4.1.2 Adjustment

The set of expectations of a society are adjusted when it acquires new knowledge from the environment on social events that was not available at the time of introducing such expectations.

Example 4.2 *Let us suppose, given the history of the example 4.1, at time 2 another event happens:*

$$H(tell(svs, f, inform(train\_info,' departs(sv,rm,10{:}15)'), d_1), 2).$$

This event fulfills one (the first) of the two alternative expectations of example 4.1. Moreover, because of IC_2, the event causes a new expectation to be generated. The adjusted set of expectations is:[13]

[12] As in example 3.1, f is Francisco's computee and svs is the San Vincenzo's station computee.

[13] In fact, $EXP(2)$ contains more disjunctions, for instance another option is to assume $E(tell(svs, f, inform(train\_info, Info), d_1), T')$: $T' \neq 2$; but in order to keep these examples simple, here and elsewhere we do not show all the possible alternative sets of expectations generated by a given course of events.

$EXP(2)$:
{

$$(\star E(tell(svs, f, inform(train\_info, Info), d_1), T_1) : T_1 < 11 \quad \vee$$
$$E(tell(svs, f, refuse(train\_info), d_1), T_1) : T_1 < 11),$$
$$EN(tell(svs, f, refuse(train\_info), d_1), T_2) : T_2 > 2.$$

}

Note that the first of the two alternative expectations is marked with a $\star$ to indicate that it is fulfilled.

4.2 OPENNESS

4.2.1 Semi-open society

According to [Dav01], societies can be classified into 4 groups, each characterized by a different degree of openness. In the following, we give an example of how our framework can model a semi-open society, i.e., a society that can be joined by a computee by executing an entering protocol. This situation manifests itself in the San Vincenzo scenario when, for example, Francisco enters the train station and buys a ticket (effectively registers himself and his computee as a member of the "train station" society).

We suppose that a particular computee indicated by constant *gatekeeper* is in charge of receiving joining requests, and requires, for computees to enter, some form to be filled. In the example of Francisco entering the train station, the gatekeeper computee is the computee that represents the train station manager.

The entering protocol is defined by the following constraints:

IC_1:

$$H(tell(C, gatekeeper, ask(register), D), T) \quad \Rightarrow$$
$$E(tell(gatekeeper, C, ask(form), D), T_1) : T_1 < T + 10$$

IC_2:

$$H(tell(C, gatekeeper, ask(register), D), T) \quad \wedge$$
$$H(tell(gatekeeper, C, ask(form), D), T_1) \wedge T < T_1 \quad \Rightarrow$$
$$E(tell(C, gatekeeper, send(form, F), D), T_2) : T_2 < T_1 + 10$$

IC_3:

$$H(tell(gatekeeper, C, ask(form), D), T_1) \quad \wedge$$
$$H(tell(C, gatekeeper, send(form, F), D), T_2) \wedge T_1 < T_2 \quad \Rightarrow$$
$$E(tell(gatekeeper, C, accept(register), D), T_3) : T_3 < T_2 + 10 \quad \vee$$
$$E(tell(gatekeeper, C, reject(register), D), T_3) : T_3 < T_2 + 10$$

In the following we assume that a computee cannot leave the society once it has become a member (although we are aware that this is a strong assumption to make, we make it here because it simplifies the presentation of the example considerably).

The presence in the history of an event of type:

$H(tell(gatekeeper, C, accept(register), D), T)$

represents the "full membership" of computee C in the society, and can be used in social integrity constraints as a condition for generating expectations.

For instance, the protocol of example 4.1 can be modified as follows to take membership into account:

IC_4:

$$
\begin{array}{ll}
H(tell(A, B, request(R), D), T) \wedge resource(R) & \wedge \\
H(tell(gatekeeper, A, accept(register), D_0), T_a) \wedge T_a < T & \wedge \\
H(tell(gatekeeper, B, accept(register), D_1), T_b) \wedge T_b < T & \Rightarrow \\
\quad E(tell(B, A, accept(R), D), T_1) : T_1 < T + 10 & \vee \\
\quad E(tell(B, A, refuse(R), D), T_1) : T_1 < T + 10 &
\end{array}
$$

IC_4 shows that, before receiving an *accept(register)*, computee A is not a member of the society in a full sense, in that A's request to B does not generate an expectation for B to answer *accept* or *refuse*.

IC_5:

$$
\begin{array}{ll}
H(tell(A, B, accept(R), D), T) & \wedge \\
H(tell(gatekeeper, A, accept(register), D_0), T_a) \wedge T_a < T & \wedge \\
H(tell(gatekeeper, B, accept(register), D_1), T_b) \wedge T_b < T & \Rightarrow \\
\quad EN(tell(A, B, refuse(R), D), T_1) : T_1 > T &
\end{array}
$$

It is apparent that this mechanism makes social integrity constraints bigger and less readable; however, it is quite easy to devise some kind of syntactic sugar to avoid the problem.

Example 4.3 *Let us consider the evolution of the EXP set for the following course of events:*

$Time01 : H(tell(f, gatekeeper, ask(register), d_1), 1).$

$EXP(01)$:

{

$E(tell(gatekeeper, f, ask(form), d_1), T_1) : T_1 < 11$

}

$Time06 : H(tell(gatekeeper, f, ask(form), d_1), 6).$

$EXP(06)$:

{

$\star E(tell(gatekeeper, f, ask(form), d_1), T_1) : T_1 < 11$

$E(tell(f, gatekeeper, send(form, F), d_1), T_2) : T_2 < 16$

}

$Time08 : H(tell(f, gatekeeper, send(form, qualification(has\_driving\_license)), d_1), 08).$

$EXP(08)$:

{

$\star E(tell(gatekeeper, f, ask(form), d_1), T_1) : T_1 < 11$

$\star E(tell(f, gatekeeper, send(form, F), d_1), T_2) : T_2 < 16$

$(E(tell(gatekeeper, f, accept(register), d_1), T_3) : T_3 < 18 \quad \vee$

$E(tell(gatekeeper, f, reject(register), d_1), T_3) : T_3 < 18)$

}

$Time17 : H(tell(gatekeeper, f, accept(register, s), d_1), 17).$

$EXP(17)$:

{

$\star E(tell(gatekeeper, f, ask(form), d_1), T_1) : T_1 < 11$

$\star E(tell(f, gatekeeper, send(form, F), d_1), T_2) : T_2 < 16$

$(\star E(tell(gatekeeper, f, accept(register), d_1), T_3) : T_3 < 18 \quad \vee$

$E(tell(gatekeeper, f, reject(register), d_1), T_3) : T_3 < 18)$

}

From time 17 onwards, f is a "full" member of the society.

Let us consider the following communicative acts inside the society (we do not report the expectations regarding the entering protocol and assume that svs is already member of the society):

Time $13 : H(tell(f, svs, query\text{-}ref(train\_info), d_{22}), 13)$.
$EXP(13)$*:*
$\{$
$\}$

The set of expectation remains empty because at time 13 f is not yet a "full" member of the society (which reflects in the body of IC_4 not being true). Instead, at time 19:

Time $19 : H(tell(f, svs, query\text{-}ref(train\_info), d_{23}), 19)$.
$EXP(19)$*:*
$\{$

$$E(tell(svs, f, inform(train\_info, Info), d_{23}), T_1) : T_1 < 29 \quad \vee$$
$$E(tell(svs, f, refuse(train\_info), d_{23}), T_1) : T_1 < 29$$

$\}$

because the body of IC_4 is now true.

4.3 ADAPTABILITY

4.3.1 Reactivity

The society should be able to react, for instance, to a non-compliant behavior of computees.

Example 4.4 *Let us consider example 4.2 again and suppose that, at time 5, the following event happens:*

$H(tell(svs, f, refuse(train\_info), d_1), 2)$.

The society is able to detect that this event is not compliant to interaction protocols, because it violates expectation

$EN(tell(svs, f, refuse(train\_info), d_1), T_1) : T_1 > 2$.

4.3.2 Adjustment

Example 4.3 can as well be interpreted as a demonstration of adjustment, in that at time 13 the society does not react to f's request, while after f becomes a member, the society reacts by generating some expectation (see time 19).

4.4 HETEROGENEITY

4.4.1 Heterogeneity(1)

One of the main characteristics of our approach is that it makes no assumptions about the internal structure, mental state or policies of interacting computees. Social integrity constraints are meant to constrain computees' *observable* behaviour rather than their internals. In fact, it is not even required that computees can be modelled as in [KMS+03], as long as they can issue communicative acts.

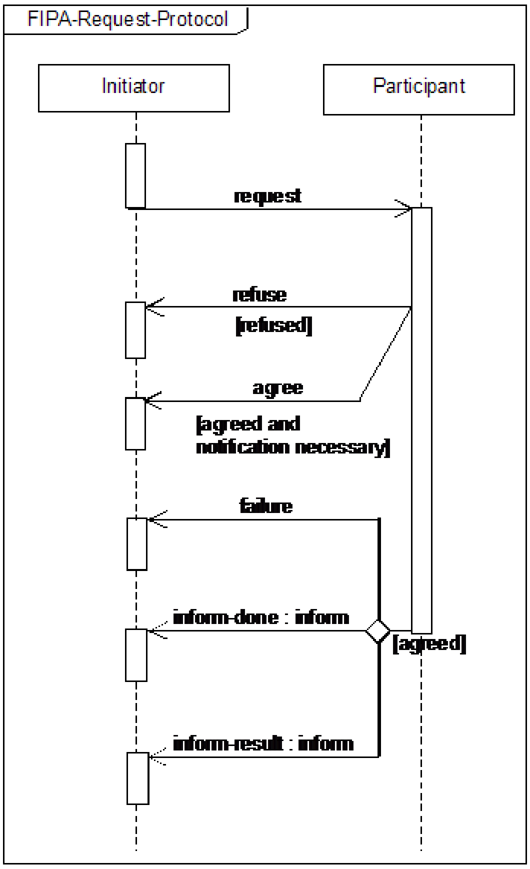

Figure 1: FIPA Request Interaction Protocol (from [FIP02])

This characteristic lets our model support both the types of heterogeneity described in [LMM$^+$03]: type 1 (heterogeneous computees as far as knowledge, reasoning capabilities and cycle) and type 2 (members of societies may have any kind of internal structure).

Thus, all of the examples presented in this section can be viewed as examples of support of both types of heterogeneity. However, in the following we present one further example: the FIPA Request Interaction Protocol [FIP02], to show that it is possible to make computees (or, more in general, agents) with different internal specifications (e.g., FIPA-compliant) successfully communicate in a SOCS society.

The protocol is depicted in Fig. 1. In the following, we specify the protocol by means of social integrity constraints.

IC_1:

$$\begin{array}{ll} H(tell(Initiator, Participant, request(P), D), T) & \Rightarrow \\ \quad E(tell(Participant, Initiator, agree(P), D), T_1) : T < T_1 & \vee \\ \quad E(tell(Participant, Initiator, refuse(P), D), T_1) : T < T_1 & \end{array}$$

IC_2:

$$
\begin{array}{ll}
H(tell(Participant, Initiator, agree(P), D), T_1) & \Rightarrow \\
\quad EN(tell(Participant, Initiator, refuse(P), D), T_2) : T_1 < T_2 &
\end{array}
$$

IC_3:

$$
\begin{array}{ll}
H(tell(Participant, Initiator, refuse(P), D), T_1) & \Rightarrow \\
\quad EN(tell(Participant, Initiator, agree(P), D), T_2) : T_1 < T_2 &
\end{array}
$$

IC_4:

$$
\begin{array}{ll}
H(tell(Initiator, Participant, request(P), D), T) & \wedge \\
H(tell(Participant, Initiator, agree(P), D), T_1) : T < T_1 & \Rightarrow \\
\quad E(tell(Participant, Initiator, failure(P), D), T_2) : T_1 < T_2 & \vee \\
\quad E(tell(Participant, Initiator, inform\text{-}done(P), D), T_2) : T_1 < T_2 & \vee \\
\quad E(tell(Participant, Initiator, inform\text{-}result(P, R), D), T_2) : T_1 < T_2 &
\end{array}
$$

IC_5:

$$
\begin{array}{ll}
H(tell(Participant, Initiator, failure(P), D), T_1) & \Rightarrow \\
\quad EN(tell(Participant, Initiator, inform\text{-}done(P), D), T_2) : T_1 < T_2 & \wedge \\
\quad EN(tell(Participant, Initiator, inform\text{-}result(P, R), D), T_2) : T_1 < T_2 &
\end{array}
$$

IC_6:

$$
\begin{array}{ll}
H(tell(Participant, Initiator, inform\text{-}done(P), D), T_1) & \Rightarrow \\
\quad EN(tell(Participant, Initiator, failure(P), D), T_2) : T_1 < T_2 & \wedge \\
\quad EN(tell(Participant, Initiator, inform\text{-}result(P, R), D), T_2) : T_1 < T_2 &
\end{array}
$$

IC_7:

$$
\begin{array}{ll}
H(tell(Participant, Initiator, inform\text{-}result(P, R), D), T_1) & \Rightarrow \\
\quad EN(tell(Participant, Initiator, failure(P), D), T_2) : T_1 < T_2 & \wedge \\
\quad EN(tell(Participant, Initiator, inform\text{-}done(P), D), T_2) : T_1 < T_2 &
\end{array}
$$

Example 4.5 *A possible realization of the protocol is the following: computee i asks computee p to book a room in a hotel; p accepts the requests, performs the action and informs i that the room is booked for 100\$.*

Time 3: $H(tell(i, p, request(book(hotel)), d), 3)$

$EXP(3)$:

$$
\left\{
\begin{array}{ll}
E(tell(p, i, agree(book(hotel)), d), T_1) : 3 < T_1 & \vee \\
E(tell(p, i, refuse(book(hotel)), d), T_1) : 3 < T_1 &
\end{array}
\right\}
$$

Time 5: $H(tell(p, i, agree(book(hotel)), d), 5)$

$EXP(5)$:

$$
\left\{
\begin{array}{ll}
(\star E(tell(p, i, agree(book(hotel)), d), T_1) : 3 < T_1 & \vee \\
E(tell(p, i, refuse(book(hotel)), d), T_1) : 3 < T_1) & \\
(E(tell(p, i, failure(book(hotel)), d), T_2) : 5 < T_2 & \vee \\
E(tell(p, i, inform\text{-}done(book(hotel)), d), T_2) : 5 < T_2 & \vee \\
E(tell(p, i, inform\text{-}result(book(hotel), R), d), T_2) : 5 < T_2) &
\end{array}
\right\}
$$

Time 10: $H(tell(i, p, inform\text{-}result(book(hotel), 100), d), 10)$

$EXP(10)$:

{

$$(\star E(tell(p,i,agree(book(hotel)),d),T_1):3<T_1 \quad \vee$$
$$E(tell(p,i,refuse(book(hotel)),d),T_1):3<T_1)$$
$$(E(tell(p,i,failure(book(hotel)),d),T_2):5<T_2 \quad \vee$$
$$E(tell(p,i,inform\text{-}done(book(hotel)),d),T_2):T_1<T_2 \quad \vee$$
$$\star E(tell(p,i,inform\text{-}result(book(hotel),R),d),T_2):5<T_2)$$
$$EN(tell(p,i,failure(book(hotel)),d),T_3):10<T_3$$
$$EN(tell(p,i,inform\text{-}done(book(hotel)),d),T_3):10<T_3$$

}

4.4.2 Heterogeneity(2)

Another kind of heterogeneity is that of societies. We can have heterogeneous societies, as well as heterogeneous computees. For instance, a society could define a protocol in a more or less flexible way. For instance, in the definition of the *Query* Protocol that we have shown in Section 4.1.1, IC_1, IC_2 and IC_3 do not prevent a computee from saying *inform* without being asked. In a way, we implement a relaxed version of the Query protocol. In a stricter society, we could have an additional (backward) constraint, like the following:

IC_4:

$$H(tell(A,B,inform(Info,Answer),D),T) \quad \Rightarrow$$
$$E(tell(A,B,query\text{-}ref(Info),D),T_1):T_1<T$$

5 Examples for WP1 and WP2: computees within societies

In this section we give two concrete examples illustrating the behaviour of individual computees (WP1) within societies, as well as the behaviour of the society (WP2) within which the computee operate. Both examples are formulated within the "Leaving San Vincenzo" scenario. The first example illustrates a situation where two computees need to exchange information. The second examples illustrate the behaviour of computees engaged within a simple auction.

5.1 Example 1: request of information

There are two computees, Francisco's computee f and the San Vincenzo train station manager svs. f requests info about the arrival of the train to Rome from svs.

5.1.1 Society

The communication protocol regulating requests is a variant of the one given in section 4.1.1, namely:

IC_1:

$$H(tell(A,B,query\text{-}ref(Info),D),T) \quad \Rightarrow$$
$$E(tell(B,A,inform(Info,Answer),D),T_1):T_1<T+10 \quad \vee$$
$$E(tell(B,A,refuse(Info),D),T_1):T_1<T+10$$

IC_2:

$$H(tell(A,B,inform(Info,Answer),D),T) \Rightarrow$$
$$EN(tell(A,B,refuse(Info),D),T_1) : T_1 > T$$

IC_3:

$$H(tell(A,B,refuse(Info),D),T) \Rightarrow$$
$$EN(tell(A,B,inform(Info,Answer),D),T_1) : T_1 > T$$

We will see how compliance to this protocol is checked by the society compliance verifier in the two settings below. In the first setting, the train manager gives an appropriate answer in time, and the social compliance verifier correctly raises no violation. In the second setting, the train manager first refuses and then gives an answer, and the verifier correctly raises a violation.

5.1.2 Setting 1

- **State of svs and f**
  - KB^{svs}_{react}:
    $observed(C, tell(C, svs, query\_ref(Q), D, T0), T)$
    $\Rightarrow assume\_happens(tell(svs, C, refuse(Q), D), T') \wedge T' > T$
  - $Plan^{svs}$, $Goals^{svs}$, $Goals^{f}$ are empty.
  - $Plan^{f}$ consists of the (reactive) action:
    $\langle tell(f, svs, query\_ref(arrival\_time(tr123)), d, T0), \perp, true, \{\}\rangle$

    for some d.
  - KB^{svs}_0 and KB^{f}_0 are empty to start with.
  - all other components of KB^{svs} and KB^{f} do not play any role here and can be considered to be empty.
- **Behaviour of svs and f**
  - Via AE at time 3, f executes the action in $Plan^{f}$, adding to KB^{f}_0
    $executed(tell(f, svs, query\_ref(arrival\_time(tr123)), d, T0), 3)$
  - Via POI at time 6, svs records the observation
    $observed(f, tell(f, svs, query\_ref(arrival\_time(tr123)), d), 3), 6)$
  - Via RE at time 10, svs adds to its $Plan$ the (reactive) action
    $\langle tell(svs, f, refuse(arrival\_time(tr123)), d, T), \perp, true, \{T > 10\}\rangle$
  - This is then executed at time 10 via AE.
  - Finally, PR clears the state.
- **Society computation**

  After the first action (at time 3), the HAP and EXP sets for the society are as follows:

  $HAP(3)$:

  {

  $H(tell(f, svs, query\text{-}ref(arrival\_time(tr123)), d), 3).$

  }

$EXP(3)$:

$$\{$$
$$E(tell(svs, f, inform(arrival\_time(tr123), Info), d_1), T_1) : T_1 < 13 \quad \vee$$
$$E(tell(svs, f, refuse(arrival\_time(tr123)), d_1), T_1) : T_1 < 13$$
$$\}$$

After the second event (at time 10), the HAP and EXP sets are the following:

$HAP(10)$:

$$\{$$
$$H(tell(f, svs, query\text{-}ref(arrival\_time(tr123)), d), 3).$$
$$H(tell(svs, f, refuse(arrival\_time(tr123)), d), 10).$$
$$\}$$

$EXP(10)$:

$$\{$$
$$(E(tell(svs, f, inform(arrival\_time(tr123), Info), d_1), T_1) : T_1 < 13 \quad \vee$$
$$\star E(tell(svs, f, refuse(arrival\_time(tr123)), d_1), T_1) : T_1 < 13),$$
$$EN(tell(svs, f, inform(arrival\_time(tr123), Info), d_1), T_2) : T_2 > 10.$$
$$\}$$

Correctly, the society does not detect any violation.

5.1.3 Setting 2

- **Computee State**

  KB^{svs}_{react}, in addition to the reactive rule for the first setting, contains the reactive rule and logic programming rules:

$$observed(C, tell(C, svs, query\_ref(Q), D, T0), T), holds\_at(have\_info(Q, I), T)$$
$$\Rightarrow assume\_happens(tell(svs, C, inform(Q, I), D), T') \wedge T' > T$$
$$precondition(tell(svs, C, inform(Q, I), D), have\_info(Q, I))$$
$$holds\_initially(have\_info(arrival\_time(tr123, 10 : 32am)))$$
$$executable(tell(svs, C, S, D)) \leftarrow C \neq svs$$
$$assume\_happens(A, T), not\, executable(A) \Rightarrow false$$

- **Computee Behaviour**

  Similarly to the earlier example, after an AE by f and a POI by svs, svs will apply RE. However, in this setting, RE will generate two *tell* actions (one for $refuse$ and one for $inform$), as well as subgoal to achieve the precondition of the $inform$ action. Then, via (one or more) AE, svs will execute them both (possibly at different times, if within two different AE transitions).

- **Society Computation**

  In addition to the events of the previous example, in this setting one more action (*inform*) is performed (say, at time 12):

$HAP(12)$:
{

$$H(tell(f, svs, \textit{query-ref}(\textit{arrival\_time(tr123)}), d), 3).$$
$$H(tell(svs, f, \textit{refuse}(\textit{arrival\_time(tr123)}), d), 10).$$
$$H(tell(svs, f, \textit{inform}(\textit{arrival\_time(tr123)}, \textit{1032}), d), 12).$$

}

$EXP(12)$:
{

$$(\star E(tell(svs, f, \textit{inform}(\textit{arrival\_time(tr123)}, \textit{Info}), d_1), T_1) : T_1 < 13 \quad \vee$$
$$\star E(tell(svs, f, \textit{refuse}(\textit{arrival\_time(tr123)}), d_1), T_1) : T_1 < 13),$$
$$\star EN(tell(svs, f, \textit{inform}(\textit{arrival\_time(tr123)}, \textit{Info}), d_1), T_2) : T_2 > 10,$$
$$EN(tell(svs, f, \textit{refuse}(\textit{arrival\_time(tr123)}), d_1), T_2) : T_2 > 12.$$

}

The society will then detect the violation of the expectation $EN(tell(svs, f, \textit{inform}(\textit{arrival\_time(tr123)}, \textit{Info}), d_1), T_2) : T_2 > 10$.

5.2 Example 2: "first price sealed bid" auction

Francisco wants a taxi to go to the station. In order to get a taxi with the lowest fare, Francisco's computee f decides to hold an auction of the kind "first price sealed bid", whereby it collects all the bids from taxi's computees first, and then notifies the winner (namely the taxi offering the lowest fare) as well as all the other bidders (taxis).

5.2.1 Society

The protocol regulating the "first price sealed bid" auction can be represented by the following social integrity constraints.

IC_{auc-1}:

$$\begin{array}{l} H(tell(B, A, bid(Item, Quote), AuctionId), T_{Bid}) \quad \Rightarrow \\ \quad E(tell(A, B, openauction(Item, T_{End}, T_{Deadline}), AuctionId), T_{Open}) : \\ \quad T_{Open} < T_{Bid},\ T_{Bid} \leq T_{End}, T_{End} < T_{Deadline} \end{array}$$

IC_{auc-2}:

$$\begin{array}{l} H(tell(A, B, openauction(Item, T_{End}, T_{Deadline}), AuctionId), T_{Open}), \\ H(tell(B, A, bid(Item, Quote), AuctionId), T_{Bid}) : \\ T_{Open} < T_{Bid}, T_{Bid} \leq T_{End}, T_{End} < T_{Deadline} \Rightarrow \\ \quad E(tell(A, B, answer(win, Item, Quote), AuctionId), T_{Win}) : \\ \quad T_{Win} < T_{Deadline}, T_{End} < T_{Win} \quad \vee \\ \quad E(tell(A, B, answer(lose, Item, Quote), AuctionId), T_{Lose}) : \\ \quad T_{Lose} < T_{Deadline}, T_{End} < T_{Lose} \end{array}$$

IC_{auc-3}:

$$\begin{array}{l} H(tell(A, B, answer(win, Item, Quote), AuctionId), T_{Win}), \quad \Rightarrow \\ \quad EN(tell(A, B, answer(lose, Item, Quote), AuctionId), T_{Lose}) : \\ \quad T_{Lose} > T_{Win} \end{array}$$

IC_{auc-4}:

$$H(tell(A, B, answer(lose, Item, Quote), AuctionId), T_{Lose}), \Rightarrow$$
$$EN(tell(A, B, answer(win, Item, Quote), AuctionId), T_{Win}) :$$
$$T_{Win} > T_{Lose}$$

5.2.2 Computee state: bidders (taxis)

$KB^{t_1}_{react}$, $KB^{t_2}_{react}$ and $KB^{t_3}_{react}$ all contain:

$$observed(tell(A, t_i, openauction(taxi, TEnd, TDeadline), Id, TOpen),$$
$$holds\_at(cost\_taxi, C), TOpen)$$
$$\Rightarrow assume\_happens(tell(t_i, A, bid(I, C), Id), T) \wedge TOpen < T < TEnd$$

Moreover, each of the taxi compute will have a concrete cost in their KB, e.g. $KB^{t_1}_{react}, KB^{t_2}_{react}, KB^{t_3}_{react}$ might have, respectively:

$$holds\_initially(cost\_taxi, 3)$$
$$holds\_initially(cost\_taxi, 5)$$
$$holds\_initially(cost\_taxi, 7)$$

5.2.3 Computee state: auctioneer (f)

In formulating the knowledge base of f, we will assume the existence of an *external object* that we will refer to as ar (standing for *auction reasoner*) which f will communicate with in order to get a solution to the (simple) optimisation problem it has to solve in order to decide the winner of the auction that f will open. We will not make any assumptions here on the internal structure of ar, but we will assume that it is "wrapped" in such a way that f can communicate with it. Moreover, we will assume that f and ar are syncronised, in the sense of sharing the same clock, and that ar is committed to return a solution of the optimisation problem by the auction deadline that f passes to it.

The following is the set of reactive rules in KB^{f}_{react}:

$$holds\_at(want\_taxi, T),\ holds\_at(available\_taxi(X), T)$$
$$\Rightarrow assume\_happens(tell(f, X, openauction(taxi, T + 5, T + 10), d(fpsb, T)), T)$$

namely, whenever in need of a taxi, f opens an auction for a taxi, communicating to all taxi drivers it believes to be available the end time of the auction ($T + 5$) and the deadline or deciding the winner of the auction ($T + 10$).

$$observed(C, tell(C, f, bid(I, C), Id, T0), T),$$
$$executed(f, tell(f, C, openauction(I, TEnd, TDeadline), Id, T), T1), T1 < T0$$
$$\Rightarrow assume\_holds(tell(f, ar, calculate(Id, I, TEnd, TDeadline), d(ar, T')), T')$$
$$\wedge T < T' < TDeadline - 2$$

namely, f send every bid it receives to the auction reasoner ar, as soon as it receives it.

$$observed(ar, tell(ar, f, results(Id, I, TEnd, TDeadline, (C, Q), R), T0), T)$$
$$executed(f, tell(f, X, openauction(I, TEnd, TDeadline), Id, T), T1), T1 < T0$$
$$\Rightarrow assume\_holds(tell(f, C, answer(win, I, Q), Id), T') \wedge T' < TDeadline$$

namely, f notifies the winner of the auction as soon as it receives the decision from ar (we assume that ar sends the notification by the deadline of the auction and thus f is able to meet this deadline in turn).

$$
\begin{aligned}
&observed(ar, tell(auction\_reasoner, f, results(Id, I, TEnd, TDeadline, (C, Q), R), T0), T) \\
&executed(f, tell(f, X, openauction(I, TEnd, TDeadline), Id), T), T1, T1 < T0 \\
&\qquad \Rightarrow assume\_happens(tell(f, C, answer(lose, I, Q), Id), T) \wedge TEnd < T < TDeadline
\end{aligned}
$$

namely, f notifies the losers of the auction as soon as it receives the decision from ar.

The following is the logic program part of KB^{f}_{react}

$holds\_initially(available(t_1))$
$holds\_initially(available(t_2))$
$holds\_initially(available(t_3))$

The following could be seen as a possible template for implementing the external object ar that performs the calculation of the best fare from a number of bids that it receives.

step 1: $findall\_bids(Id, I, TEnd, S)$,
step 2: $find\_minimum(S, (C, Q))$,
step 3: $R = minus(S, (C, Q))$

5.2.4 Computees' behaviours

The auctioneer (f) generates (via RE) and executes (via AE) an action to open the auction. Then the bidders (taxis) record the communicative act via POI. Further, they put forward their bids (offering fares of 3, 5, 7, respectively), again by RE and AE. Then, via POI, f records the bids and, via RE, it sends them to the ar. After having received the final calculations from ar, recorded via POI, f generates, via RE, the communicative actions to make t_1 a winner and the other two losers, and, via AE, it executes the actions. In the background, the society checks for conformance to the protocols, and finds no violation (see section 5.2.5, first setting).

If the last reactive rule of f is missing, namely the losers are not notified, then the society detects a violation instead (see section 5.2.5, second setting).

Note that the external object ar is not a proper component of the society, and thus the society does not check the conformance of its behaviour to protocols (actually, there is no society integrity constraint about ar). Thus, computees may be declared as non-conformant to the given protocols even when in reality it is the external object ar which is non-conformant (e.g. by giving the final calculations of the winner to f after the deadline).

5.2.5 Society Computations

- **First Setting**

  After the *openauction* actions, the HAP set will be the following:

  $HAP(5)$

  $$
  \{ \\
  H(tell(f, taxi1, openauction(taxi, 10, 20), auction1), 5). \\
  H(tell(f, taxi2, openauction(taxi, 10, 20), auction1), 5). \\
  H(tell(f, taxi3, openauction(taxi, 10, 20), auction1), 5). \\
  \}
  $$

No expectations are generated so far.

After the three *bid* actions, the HAP and EXP sets are as follows:

$HAP(9)$

$\{$
$H(tell(f, taxi1, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(f, taxi2, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(f, taxi3, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(taxi1, f, bid(taxi, 3), auction1), 7).$
$H(tell(taxi2, f, bid(taxi, 5), auction1), 8).$
$H(tell(taxi3, f, bid(taxi, 7), auction1), 9).$
$\}$

$EXP(9)$

$\{$
$\star E(tell(f, taxi1, openauction(taxi, T_{End1}, T_{Deadline1}), auction1), T_1) :$
$T_1 < 7,\ 7 \leq T_{End1},\ T_{End1} < T_{Deadline1}.$
$\star E(tell(f, taxi2, openauction(taxi, T_{End2}, T_{Deadline2}), auction1), T_2) :$
$T_2 < 8,\ 8 \leq T_{End2},\ T_{End2} < T_{Deadline2}.$
$\star E(tell(f, taxi3, openauction(taxi, T_{End3}, T_{Deadline3}), auction1), T_3) :$
$T_3 < 9,\ 9 \leq T_{End3},\ T_{End3} < T_{Deadline3}.$
$(E(tell(f, taxi1, answer(win, taxi, 3), auction1), T_{Win1}) : T_{Win1} < 20, 10 < T_{Win1} \vee$
$E(tell(f, taxi1, answer(lose, taxi, 3), auction1), T_{Lose1}) : T_{Lose1} < 20, 10 < T_{Lose1})$
$(E(tell(f, taxi2, answer(win, taxi, 5), auction1), T_{Win2}) : T_{Win2} < 20, 10 < T_{Win2} \vee$
$E(tell(f, taxi2, answer(lose, taxi, 5), auction1), T_{Lose2}) : T_{Lose2} < 20, 10 < T_{Lose2})$
$(E(tell(f, taxi3, answer(win, taxi, 7), auction1), T_{Win3}) : T_{Win3} < 20, 10 < T_{Win3} \vee$
$E(tell(f, taxi3, answer(lose, taxi, 7), auction1), T_{Lose3}) : T_{Lose3} < 20, 10 < T_{Lose3})$
$\}$

The first three expectations (generated because of IC_{auc-1}) are already fulfilled, while the others (generated because of IC_{auc-2}) are not.

$HAP(14)$

$\{$
$H(tell(f, taxi1, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(f, taxi2, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(f, taxi3, openauction(taxi, 10, 20), auction1), 5).$
$H(tell(taxi1, f, bid(taxi, 3), auction1), 7).$
$H(tell(taxi2, f, bid(taxi, 5), auction1), 8).$
$H(tell(taxi3, f, bid(taxi, 7), auction1), 9).$
$H(tell(f, taxi1, answer(win, taxi, 3), auction1), 12).$
$H(tell(f, taxi2, answer(lose, taxi, 5), auction1), 13).$
$H(tell(f, taxi3, answer(lose, taxi, 7), auction1), 14).$
$\}$

$EXP(14)$

$$
\begin{array}{l}
\{ \\
\quad \star E(tell(f, taxi1, openauction(taxi, T_{End1}, T_{Deadline1}), auction1), T_1) : \\
\quad T_1 < 7,\ 7 \leq T_{End1},\ T_{End1} < T_{Deadline1}. \\
\quad \star E(tell(f, taxi2, openauction(taxi, T_{End2}, T_{Deadline2}), auction1), T_2) : \\
\quad T_2 < 8,\ 8 \leq T_{End2},\ T_{End2} < T_{Deadline2}. \\
\quad \star E(tell(f, taxi3, openauction(taxi, T_{End3}, T_{Deadline3}), auction1), T_3) : \\
\quad T_3 < 9,\ 9 \leq T_{End3},\ T_{End3} < T_{Deadline3}. \\
\quad (\star E(tell(f, taxi1, answer(win, taxi, 3), auction1), T_{Win1}) : T_{Win1} < 20, 10 < T_{Win1} \vee \\
\quad E(tell(f, taxi1, answer(lose, taxi, 3), auction1), T_{Lose1}) : T_{Lose1} < 20, 10 < T_{Lose1}) \\
\quad (E(tell(f, taxi2, answer(win, taxi, 5), auction1), T_{Win2}) : T_{Win2} < 20, 10 < T_{Win2} \vee \\
\quad \star E(tell(f, taxi2, answer(lose, taxi, 5), auction1), T_{Lose2}) : T_{Lose2} < 20, 10 < T_{Lose2}) \\
\quad (E(tell(f, taxi3, answer(win, taxi, 7), auction1), T_{Win3}) : T_{Win3} < 20, 10 < T_{Win3} \vee \\
\quad \star E(tell(f, taxi3, answer(lose, taxi, 7), auction1), T_{Lose3}) : T_{Lose3} < 20, 10 < T_{Lose3}) \\
\quad EN(tell(f, taxi1, answer(lose, taxi, 3), auction1), T_{Lose1}) : T_{Lose1} > 12 \\
\quad EN(tell(f, taxi2, answer(win, taxi, 5), auction1), T_{Win2}) : T_{Win2} > 13 \\
\quad EN(tell(f, taxi3, answer(win, taxi, 7), auction1), T_{Win3}) : T_{Win3} > 14 \\
\}
\end{array}
$$

One branch of each of the three ORs has been fulfilled by the subsequent three events, and the three negative expecations about the "contrary" notification actions have been generated because of IC_{auc-3} and IC_{auc-4}.

If the history is closed here, no expectations will be violated.

- **Second Setting**

In this case, missing the two *lose* notifications, the final HAP and EXP will be:

$HAP(14)$

$$
\begin{array}{l}
\{ \\
\quad H(tell(f, taxi1, openauction(taxi, 10, 20), auction1), 5). \\
\quad H(tell(f, taxi2, openauction(taxi, 10, 20), auction1), 5). \\
\quad H(tell(f, taxi3, openauction(taxi, 10, 20), auction1), 5). \\
\quad H(tell(taxi1, f, bid(taxi, 3), auction1), 7). \\
\quad H(tell(taxi2, f, bid(taxi, 5), auction1), 8). \\
\quad H(tell(taxi3, f, bid(taxi, 7), auction1), 9). \\
\quad H(tell(f, taxi1, answer(win, taxi, 3), auction1), 12). \\
\}
\end{array}
$$

$EXP(14)$

$$
\begin{array}{l}
\{ \\
\quad \star E(tell(f, taxi1, openauction(taxi, T_{End1}, T_{Deadline1}), auction1), T_1) : \\
\quad T_1 < 7,\ 7 \leq T_{End1},\ T_{End1} < T_{Deadline3}. \\
\quad \star E(tell(f, taxi2, openauction(taxi, T_{End2}, T_{Deadline2}), auction1), T_2) : \\
\quad T_2 < 8,\ 8 \leq T_{End2},\ T_{End2} < T_{Deadline3}. \\
\quad \star E(tell(f, taxi3, openauction(taxi, T_{End3}, T_{Deadline3}), auction1), T_3) : \\
\quad T_3 < 9,\ 9 \leq T_{End3},\ T_{End3} < T_{Deadline3}. \\
\quad (\star E(tell(f, taxi1, answer(win, taxi, 3), auction1), T_{Win1}) : T_{Win1} < 20, 10 < T_{Win1} \vee \\
\quad E(tell(f, taxi1, answer(lose, taxi, 3), auction1), T_{Lose1}) : T_{Lose1} < 20, 10 < T_{Lose1}) \\
\quad (E(tell(f, taxi2, answer(win, taxi, 5), auction1), T_{Win2}) : T_{Win2} < 20, 10 < T_{Win2} \vee \\
\quad E(tell(f, taxi2, answer(lose, taxi, 5), auction1), T_{Lose2}) : T_{Lose2} < 20, 10 < T_{Lose2}) \\
\quad (E(tell(f, taxi3, answer(win, taxi, 7), auction1), T_{Win3}) : T_{Win3} < 20, 10 < T_{Win3} \vee \\
\quad E(tell(f, taxi3, answer(lose, taxi, 7), auction1), T_{Lose3}) : T_{Lose3} < 20, 10 < T_{Lose3}) \\
\quad EN(tell(f, taxi1, answer(lose, taxi, 3), auction1), T_{Lose1}) : T_{Lose1} > 12 \\
\}
\end{array}
$$

Both the expectations of the last two ORs remain unfulfilled: thus, closing the history causes a violation.

6 Conclusions

We have presented a number of examples for the functioning of individual computees and of societies. For each example, we have provided a detailed description of the knowledge involved, of the behaviour expected, and of the feature(s) the example is exemplifying in the context of Global Computing. We have also presented examples of the integration between societies and computees, with a detailed description of the knowledge involved and of the behaviour expected. The second integrated example requires the interaction of computees with an external object, whose internal structure may be different from that of computees, but which is assumed to be "wrapped" in such a way as to be able to communicate with computees. The example is a simple first-price-sealed-auction. We have already started investigating the modeling and realisation within the computee and society models and the SOCS prototype of more complex combinatorial auctions [MMR+02]. In the case of combinatorial auctions, we can use a more sophisticated external object for the winner determination, e.g. one based on an ILOG-solver [MMR+02], and basically adopt the same social integrity constraints and knowledge bases of computees as for the simpler first-price-sealed-auction that we have considered.

Acknowledgements

We would like to thank Nicolas Maudet and Giacomo Terreni for useful disucssions and contributions to an earlier version of this document.

References

[CS01] M. Cohen and K. Stathis. Strategic Change stemming from E-Commerce: Implications of Multi-Agent Systems in the Supply Chain. *Strategic Change*, 10:139–149, 2001.

[Dav01] P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents World II*, volume 2203 of *Lecture Notes in Artificial Intelligence*, pages 1–9. Springer-Verlag, December 2001. 2nd International Workshop (ESAW'01), Prague, Czech Republic, 7 July 2001, Revised Papers.

[EMST02] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Resource Allocation by Negotiation. Technical Report IST32530/ICSTM/019/IN/I/a2, SOCS consortium, 2002.

[FIP01a] FIPA Communicative Act Library Specification, August 2001. Published on August 10th, 2001, available for download from the FIPA website.

[FIP01b] FIPA Query Interaction Protocol, August 2001. Published on August 10th, 2001, available for download from the FIPA website.

[FIP02] FIPA Request Interaction Protocol Specification. Standard SC00026H, Foundation for Intelligent Physical Agents, December 2002. Published on December 3, 2002, available for download from the FIPA website.

[KM02] A. C. Kakas and P. Moraïtis. Argumentation based decision making for autonomous agents. Technical report, Departmento of Computer Science, University of Cyprus, 2002. Submitted to AAMAS 2003.

[KMS+03] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Deliverable D4.

[LMM+03] E. Lamma, P. Mello, P. Mancarella, A. Kakas, K. Stathis, and F. Toni. Self-assessment: parameters and criteria. Technical report, SOCS Consortium, 2003. Deliverable D3. Distribution restricted to the GC programme.

[MMR+02] P. Melo, M. Milano, A. Roli, R. Montanari, M. Gavanelli, E. Lamma, and F. Riguzzi. Combinatorial Auctions. Technical Report IST32530/UNIBO/0XX/IN/I/a2, SOCS consortium, 2002.

[MTG+03] P. Mello, P. Torroni, M. Gavanelli, M. Alberti, A. Ciampolini, M. Milano, A. Roli, E. Lamma, F. Riguzzi, and N. Maudet. A logic-based approach to model interaction amongst computees. Technical report, SOCS Consortium, 2003. Deliverable D5.

[SdM02] K. Stathis, O. deBruijn, and S. Macedo. Living memory: agent-based information managment for connected local communities. *Interacting with Computers*, 14(6):665–690, 2002.

[Sta02] K. Stathis. Location-aware SOCS: The Leaving San Vincenzo scenario. Technical Report IST32530/CITY/002/IN/PP/a1, SOCS consortium, 2002.

[STT02] F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 419–431. Springer-Verlag, September 2002.

[TS02] F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.

[WS03] M. Witkowski and K. Stathis. A dialectic architecture for computational autonomy. In Matthias Nickles and Michael Rovatsos, editors, *AUTONOMY 2003*, to appear as LNCS in 2003.

A Normal cycle theory

The *normal cycle theory* specifies a pattern of operation where the computee prefers to follow a sequence of transitions that allows it to achieve its goals in a way that matches an expected "normal" behaviour. Basically, the "normal" computee first introduces goals (if it has none to start with) via GI, then reacts to them, via RE, and then repeats the process of planning for them, via PI, executing (part of) the chosen plans, via AE, revising its state, via GR and PR, until all goals are dealt with (succesfully or revised away). At this point the computeee returns to introduce new goals via GI and repeats the process above. Whenever in this process the computee is interrupted via a passive observation, via POI, it chooses to introduce new goals via GI, to take into account any changes in the world. Whenever it has actions which are "unreliable", in the sense that their preconditions definitely need to be checked, the computee senses them (via SI) before executing the action. Whenever it has actions which are "unreliable", in the sense that their effects definitely need to be checked, the computee actively introduces actions that aim at sensing these effects, via AOI, after having executed the original actions.

Below, we give all the (domain independent) parts of the normal $\mathcal{T}_{cycle}$, excluding its auxiliary part. For more details on this theory, see deliverable D8. Note also that below we give the version of $\mathcal{T}_{cycle}$ required wrt the notion of operational trace that we presented in D4, rather than its generalised version as given in D8 (where we can also reason about whether to be interrupted or not). See D8 for more details.

- $\mathcal{T}_{behaviour}$ consists of the following rules (besides its auxiliary part, including the definitions for *incompatible* given above as well as any other definitions for predicates used in $\mathcal{T}_{behaviour}$, such as $empty\_goals, unreliable\_pre$ etc):

  - GI should be given higher priority if there are no goals in $Goals$ and actions in $Plan$ in the state: [14]

    $$R^{T}_{GI|T'} : h\_p(r_{T|GI}(S), r_{T|T'}(S,X)) \leftarrow empty\_goals(S), empty\_plan(S)$$

    for all transitions T, T', $T' \neq GI$, and with T possibly 0 (indicating that if there are no goals and plans in the initial state of a computee, then GI should be its first transition).

  - GI is also given higher priority after a POI:

    $$R^{POI}_{GI|T} : h\_p(r_{POI|GI}(S'), r_{POI|T}(S,S'))$$

    for all transitions $T \neq GI$.

[14] Instead of the conditions $empty\_goals(S), empty\_plan(S)$ in this rule, we could have these conditions in each rule in $\mathcal{T}_{basic}$ which indicates as viable any transition that could be a competitor of GI after any given transition.

– After GI, the transition RE should be given higher priority:

$$R^{GI}_{RE|T} : h\_p(r_{GI|RE}(S), r_{GI|T}(S, X))$$

for all transitions $T \neq RE$.

– After RE, the transition PI should be given higher priority:

$$R^{RE}_{PI|T} : h\_p(r_{RE|PI}(S, Gs), r_{RE|T}(S, X))$$

for all transitions $T \neq PI$;

– After PI, the transition AE should be given higher priority, unless there are actions in the actions selected for execution whose preconditions are "unreliable" and need checking, in which case SI will be given higher priority:

$$R^{PI}_{AE|T} : h\_p(r_{PI|AE}(S, As), r_{PI|T}(S, X)) \leftarrow not\, unreliable\_pre(As)$$

for all transitions $T \neq AE$;

$$R^{PI}_{SI|T} : h\_p(r_{PI|SI}(S, Ps), r_{PI|T}(S, As)) \leftarrow unreliable\_pre(As)$$

for all transitions $T \neq SI$;

Here we assume that the auxiliary part of $\mathcal{T}_{cycle}$ specifies whether a given set of actions contains any "unreliable" action, in the sense described above.

– After SI, the transition AE should be given higher priority

$$R^{SI}_{AE|T} : h\_p(r_{SI|AE}(S, As), r_{SI|T}(S, X))$$

for all transitions $T \neq AE$.

– After AE, the transition AE should be given higher priority until there are no more actions to execute in $Plan$, in which case either AOI or GR should be given higher priority, depending on whether there are actions which are "unreliable", in the sense that their effects need checking, or not:

$$R^{AE}_{AE|T} : h\_p(r_{AE|AE}(S, As), r_{AE|T}(S, X))$$

for all transitions $T \neq AE$. Note that, by definition of $\mathcal{T}_{basic}$ below, the transition AE is applicable only if there are still actions to be executed in the state.

$$R^{AE}_{AOI|T} : h\_p(r_{AE|AOI}(S, Fs), r_{AE|T}(S, X)) \leftarrow BC^{AE}_{AOI|T}(S, Fs)$$

for all transitions $T \neq AOI$, where the behaviour condition $BC^{AE}_{AOI|T}(S, Fs)$ is defined (in the auxiliary part) by:

$$BC^{AE}_{AOI|T}(S, FS) \leftarrow empty\_executable\_plan(S), unreliable\_post(S)$$

Similarly, we have:

$$R^{AE}_{GR|T} : h\_p(r_{AE|GR}(S), r_{AE|T}(S, X)) \leftarrow BC^{AE}_{GR|T}(S)$$

for all transitions $T \neq GR$ where:

$$BC^{AE}_{GR|T}(S) \leftarrow empty\_executable\_plan(S), not\, unreliable\_post(S)$$

Here, we assume that the auxiliary part of $\mathcal{T}_{cycle}$ specifies whether a given set of actions contains any "unreliable" action, in the sense expressed by $unreliable\_post$, and defines the predicate $empty\_executable\_plan$. Intuitively, $empty\_executable\_plan(S)$ succeeds if all the actions which can be selected for execution have already been executed.

– After GR, the transition PR should have higher priority:

$$R^{GR}_{PR|T} : h\_p(r_{GR|PR}(S, \_), r_{GR|T}(S))$$

for all transitions $T \neq PR$;

– After PR, the transition PI should have higher priority:

$$R^{PR}_{PI|T} : h\_p(r_{PR|PI}(S, Gs), r_{PR|T}(S))$$

for all transitions $T \neq PI$. Note that, by definition of $\mathcal{T}_{basic}$ below, the transition PI is applicable only if there are still goals to plan for in the state. If there are no actions and goals left in the state, then rule $R^{T'}_{GI|T}$ would apply.

– In the initial state, PI should be given higher priority:

$$h\_p(r_{0|PI}(S, Gs), r_{0|T}(S, X))$$

for all transitions $T \neq PI$. Note that PI, by definition of $\mathcal{T}_{initial}$ below, the transition PI is applicable initially only if there are goals to plan for in the initial state.

- $\mathcal{T}_{initial}$ consists of

$$r_{0|GI}(S_0) : GI(S_0) \leftarrow empty\_goals(S_0)$$
$$r_{0|PI}(S_0) : PI(S_0, Gs) \leftarrow Gs = c_{GS}(S_0, \tau), Gs \neq \{\}$$

- $\mathcal{T}_{basic}$ consists of the following parts.

– The rules for deciding what might follow an AE transition are as follows:

$$r_{AE|PI}(S', Gs) : PI(S', Gs) \leftarrow AE(S, As, S'), Gs' = c_{GS}(S', \tau), Gs' \neq \{\}$$
$$r_{AE|AE}(S', As') : AE(S', As') \leftarrow AE(S, As, S'), As' = c_{AS}(S', \tau), As' \neq \{\}$$
$$r_{AE|AOI}(S', Fs) : AOI(S', Fs) \leftarrow AE(S, As, S'), Fs = c_{FS}(S', \tau), Fs \neq \{\}$$
$$r_{AE|PR}(S') : PR(S') \leftarrow AE(S, S')$$
$$r_{AE|GI}(S') : GI(S') \leftarrow AE(S, S')$$

Namely, AE could be followed by another AE, or by a PI, or by an AOI, or by a PR, or by a GI, or by a POI. Any other possibility, e.g. for GR to follow AE, is excluded within this particular $\mathcal{T}_{basic}$ theory.

– The rules for deciding what might follow GR are as follows

$$r_{GR|PR}(S') : PR(S') \leftarrow GR(S, S')$$

Namely, GR can only be followed by PR. Indeed, GR and PR are naturally coupled, since removing some goals in the state might lead to removing some actions.

– The rules for deciding what might follow PR are as follows

$$r_{PR|PI}(S') : PI(S', Gs) \leftarrow PR(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$$
$$r_{PR|GI}(S') : GI(S') \leftarrow PR(S, S'), Gs = c_{GS}(S', \tau), Gs = \{\}$$

Namely, PR can only be followed by PI or GI, depending on whether there are goals to plan for or not in the state.

– The rules for deciding what might follow PI are as follows

$$r_{PI|AE}(S', As) : AE(S', As) \leftarrow PI(S, Gs, S'), As = c_{AS}(S', \tau), As \neq \{\}$$
$$r_{PI|SI}(S', Ps) : SI(S', Ps) \leftarrow PI(S, Gs, S'), Ps = c_{PS}(S', \tau), Ps \neq \{\}$$

The second rule is here to allow the possibility of sensing the preconditions of an action before its execution.

– The rules for deciding what might follow GI are as follows

$$r_{GI|RE}(S', Gs) : RE(S', Gs) \leftarrow GI(S, S')$$
$$r_{GI|PI}(S', Gs) : PI(S', Gs) \leftarrow GI(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$$

Namely, GI can only be followed by RE or PI, if there are goals to plan for.

- The rules for deciding what might follow RE are as follows

  $r_{RE|PI}(S', Gs) : PI(S', Gs) \leftarrow RE(S, S'), Gs = c_{GS}(S', \tau), Gs \neq \{\}$

  $r_{RE|SI}(S', Ps) : SI(S', Ps) \leftarrow RE(S, S'), Ps = c_{PS}(S', \tau), Ps \neq \{\}$

- The rules for deciding what might follow SI are as follows

  $r_{SI|AE}(S', As) : AE(S', As) \leftarrow SI(S, Ps, S'), As = c_{AS}(S', \tau), As \neq \{\}$

  $r_{SI|PR}(S') : PR(S') \leftarrow SI(S, Ps, S')$

- The rules for deciding what might follow AOI are as follows

  $r_{AOI|AE}(S', As) : AE(S', As) \leftarrow AOI(S, Fs, S'), As = c_{AS}(S', \tau), As \neq \{\}$

  $r_{AOI|GR}(S') : PR(S') \leftarrow AOI(S, Fs, S')$

  $r_{AOI|SI}(S', Ps) : SI(S', Ps) \leftarrow AOI(S, Fs, S')Ps = c_{PS}(S', \tau), Ps \neq \{\}$

- $\mathcal{T}_{interrupt}$ consists of the following rules

  $r_{POI|GI}(S') : GI(S') \leftarrow POI(S, S')$

  $r_{POI|RE}(S') : RE(S') \leftarrow POI(S, S')$

  $r_{POI|GR}(S') : GR(S') \leftarrow POI(S, S')$

  $r_{POI|POI}(S') : POI(S') \leftarrow POI(S, S')$

Note that this is just an example of cycle theory, and many others are possible. To obtain different patterns of behaviour we can restrict the rules in $\mathcal{T}_{basic}$ either by adding extra conditions in the body of its rules or by dropping some rules completely. For example, we could add the conditions $empty\_goals(S'), empty\_plan(S')$ in all the rules in $\mathcal{T}_{basic}$ introducing GI, namely all rules $r_{*|GI}(S', Gs)$, thus allowing GI only if the state is empty. This would in turn allow us to drop the conditions $empty\_goals(S'), empty\_plan(S')$ in the rule $R^{T}_{GI|T'}$ in $\mathcal{T}_{behaviour}$. As another example, we may want to drop the last rule in the definition of what might follow AE in $\mathcal{T}_{basic}$ as we may never want to do GI unless we have new observational input.